



ORACLE
NETSUITE

SuiteScript Code Samples



2023.1

May 10, 2023



Copyright © 2005, 2023, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

If this document is in public or private pre-General Availability status:

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

If this document is in private pre-General Availability status:

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your pre-General Availability trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document may change and 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.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <https://www.oracle.com/corporate/accessibility>.

Access to Oracle Support

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

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is in pre-General Availability status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your pre-General Availability trial agreement only. It is not a commitment to deliver any material, code, or functionality, and 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.

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

Code Samples Catalog Overview	1
SuiteScript Samples by Module	2
N/action Samples	3
Execute a Bulk Action on a Timebill Record	3
Find Actions Available for the Timebill Record Asynchronously Using Promise Methods	5
Locate and Execute an Action on a Timebill Record	6
N/auth Samples	7
Change a NetSuite Email Address and Password	7
N/cache Samples	8
Look Up Folder IDs	8
Retrieve Name of a City Based on a ZIP Code Using Cache and a Custom Loader Function	9
N/certificateControl Samples	11
Create, Modify, and Save Certificate Record Based on a File in the File Cabinet	11
Establish an SFTP Connection Using an SSH Key; Create, Update, Load, and Delete a Certificate Record	12
Filter the Digital Certificate List by Subsidiary and File Type	13
Find and Use an Existing Certificate Record	14
Find the Audit Trail of POST Operations for a Certificate Record Based on ID	14
Generate Signature of a Plain Text String and Verify the Signature Using the Same Certificate	15
N/commerce/recordView Samples	15
Retrieve Website and Item Data	15
N/compress Samples	16
Compress and Decompress a File	16
Create a ZIP File	17
N/config Samples	18
Load the Company Information Configuration Page and Set Field Values	18
N/crypto Samples	19
Create a Secure Key Using SHA512	19
Create a Suitelet to Request User Credentials, Create a Secret Key, and Encode a Sample String	20
N/crypto/certificate Samples	21
Create Signer and Verifier Objects	21
Load an XML File from the File Cabinet and Sign It Using a Digital Certificate	22
N/currency Samples	22
Obtain the Exchange Rate Between the Canadian Dollar and the U.S. Dollar	23
N/currentRecord Samples	23
Perform Field Sourcing Synchronously	23
Update Fields on Current Record	24
Use a Custom Module Client Script	25
N/dataset Samples	27
Create a Dataset, Run the Dataset, and List All Existing Datasets	27
List All Datasets and Load the First Dataset	28
N/email Samples	29
Send an Email with an Attachment	29
N/encode Samples	30
Convert a String to a Different Encoding	30
N/error Samples	31
Create a Custom Error	31
Create an Error Based on a Condition	31
N/file Samples	32
Create a File, Set Property Values, and Save It to the File Cabinet	32
Create and Save a CSV File then Reload the File and Parse Its Contents	33
Create and Save a File to the File Cabinet	34

Read and Log File Contents Using Commas and New Lines as Separators	35
Read and Log Segments of a File Using a Set of Characters as Separators	36
N/format Samples	36
Format a Number as a String	36
Format Time of Day as a String	37
Parse a String to a Date Object	37
Parse a String to a Number	38
N/format/i18n Samples	39
Format 12345 as a German String	39
Format a Number as a String Using N/format/i18n	40
Format Currency Based on the Locale Parameter	40
Format Numbers and Currencies Based on the English-India Locale Parameter	41
Format Numbers as Currency Strings	42
Format Numbers Based on the Locale Parameter	42
Parse a Czech Republic Phone Number	43
Parse a U.S. Phone Number	44
N/http Samples	44
Redirect a New Sales Order and Set the Entity Field	44
Request a URL Using http.get	45
N/https Samples	46
Concatenate API Secrets with Strings	46
Create a JWT Token Using a Secure String	46
Create a Form with a Field that Generates a GUID	48
Create an Authentication Header Using a Secure String	49
Generate a Secure Token and a Secret Key	50
N/https/clientCertificate Samples	51
Send a Secure Post Request to a Remote URL	51
N/keyControl Samples	52
Add a Secret Key Field to a Form	52
Create a Secret Key	53
N/log Samples	53
Create Debug Log Messages	53
N/piremoval Samples	54
Remove Phone Numbers and Comments from Customer Records	54
N/plugin Samples	55
Find Plug-in Implementations	55
N/portlet Samples	56
Create a Form Portlet with a Button That Allows User Adjustments	56
N/query Samples	58
Create a Query for a Custom Field	58
Create a Query for Customer Records and Run It as a Non-Paged Query	59
Create a Query for Transaction Records and Run It as a Paged Query	61
Convert a Query to a SuiteQL and Run It	62
Create a Query Using a Specific Record Field	62
Run an Arbitrary SuiteQL Query	63
N/record Samples	64
Access Sublists and a Subrecord from a Record	64
Access Sublists and a Subrecord from a Record Asynchronously Using Promise Methods	65
Call a Macro on a Sales Order Record	67
Create and Save a Contact Record	68
Create and Save a Contact Record Asynchronously Using Promise Methods	69
Create Multiple Sales Records Using a Scheduled Script	70
N/recordContext Samples	70
Get the Localization Context of an Employee Record	71
N/redirect Samples	71

Set a Redirect URL to a Newly Created Task Record	72
N/render Samples	72
Generate a PDF from a Raw XML String	72
Render a PDF	73
Render a Transaction Record Into an HTML Page	73
Render an Invoice Into a PDF Using an XML Template	74
Render Search Results Into a PDF	75
N/runtime Samples	75
Create Multiple Sales Records Using a Scheduled Script	76
Return User and Session Information	76
N/search Samples	77
Create a Search for a Custom Record Type	77
Delete a Saved Search	78
Load a Search for Sales Order Records and Return the First 100 Search Results	78
Load a Search for Sales Order Records and Use a Callback Function to Process Results	79
Load and Run a Paginated Search and Process the Results	80
Search for Customer Records and Log First 50 Results	80
Search for Items in a Custom List	81
Search for Sales Order Records	82
N/sftp Samples	82
Manage Files and Directories	82
Set Conditional Default Settings Using N/sftp Enums	85
Upload and Download a File	86
N/sso Samples	87
Generate a New OAuth Token	87
Generate a suiteSignOn Token in a Portlet	87
Generate a suiteSignOn Token Using a Suitelet	88
Generate a suiteSignOn Token Using a User Event Script	89
N/suiteAppInfo Samples	89
Retrieve Information for a SuiteApp	90
N/task Samples	90
Create and Submit a Map/Reduce Script Task	91
Create and Submit a Task with Dependent Scripts	92
Create and Submit an Asynchronous Search Task and Export the Results into a CSV File	95
Submit a Record Action Task and Check Status	96
N/task/accounting/recognition Samples	97
Merge Revenue Arrangements Using a Saved Search	97
Merge Revenue Arrangements Using an Ad-Hoc Search	98
Merge Revenue Elements Using Internal IDs	99
N/transaction Samples	99
Void a Sales Order Transaction	100
N/translation Samples	101
Access Parameterized Translation Strings	101
Access Translation Strings	102
Access Translation Strings Using a Non-Default Locale	102
Load Specific Translation Strings from a Collection	103
Load Translation Strings by Key from Multiple Translation Collections	104
Load Translation Strings by Key from a Translation Collection with Multiple Locales	105
N/ui/dialog Samples	106
Create a Confirmation Dialog	106
Create a Dialog with Buttons	107
Create a Dialog that Includes a Default Button	107
Create an Alert Dialog	108
N/ui/message Samples	109
Create Confirmation, Information, Warning, and Error Messages	109

N/ui/serverWidget Samples	110
Create a Custom Form with a Submit Button, Fields, and an Inline Editor Sublist	110
Create a Custom Survey Form	112
N/url Samples	114
Create a URL and Send a Secure HTTPS Post Request to the URL	115
Generate an Absolute URL to a Specific Resource	115
Retrieve the Domain for Calling a RESTlet	116
Retrieve the Relative URL of a Record	117
N/util Samples	117
Set Fields on a Sales Order Record Using the util.each Iterator	117
N/workbook Samples	118
Create Datasets, Dataset Links, and a Workbook with a Pivot and Run the Workbook	118
Create a Comprehensive Workbook	120
N/workflow Samples	123
Search For and Execute a Workflow Deployed on the Customer Record	123
N/xml Samples	124
Load an XML File and Obtain Child Element Values	124
Load an XML File from the File Cabinet and Sign It using a Digital Certificate	125
Modify an XML File	126
Parse an XML String and Log Element Values	126
Render an Invoice Into a PDF Using an XML Template	127
SuiteScript Use Cases Samples	129
Add a Custom Button to Execute a Suitelet	129
Calculate Commission on a Sales Order	130
Copy a Value to the Item Column	131
Set a Default Posting Period in a Custom Field	132
Track Deposits and Refunds	133
Set the Purchase Order Exchange Rate	138
Custom Plug-in Samples	140
Create a Custom Plug-in for Inbound E-Document Validation	140
Create a Custom Plug-in Implementation for E-Document Custom Data Source	141
Create a Custom Plug-in Implementation for Sending E-Documents	142
Create a Digital Signature Plug-in Implementation for E-Documents	144
Create a Quality Custom Inspection Rule Plug-in	145
Create a Script for Sending E-Documents	146
Update E-Document Certification Statuses	147

Code Samples Catalog Overview

📌 Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This SuiteScript Code Samples Catalog contains code samples that you can use for reference. You can copy and update them to fit your needs.

This catalog includes the following sample categories. Note that each sample may be included in multiple categories.

- [SuiteScript 2.1 Samples](#) – This section includes samples that use SuiteScript 2.1 code constructs.
- [SuiteScript Samples by Script Type](#) – This section includes samples specific to each SuiteScript script type.
- [SuiteScript Samples by Function](#) – This section includes samples that cover distinct SuiteScript functions and capabilities.
- [SuiteScript Samples by Module](#) – This section includes samples for each SuiteScript API module. For more information about SuiteScript modules, see the help topic [SuiteScript 2.x Modules](#)
- [SuiteScript Samples That Use Promises](#) – This section includes samples that use SuiteScript promise methods.
- [Custom Plug-in Samples](#) – This section includes custom plug-in samples. For each plug-in code sample, you can read an associated help topic that describes the full context of the sample and includes additional implementation details. For more information, see the help topic [Custom Plug-in Overview](#).
- [SuiteScript Use Cases Samples](#) – This section includes full use case samples. These codes samples are a small portion of the full SuiteScript tutorials that describe how to perform functions in a step-by-step approach. For more information, see the help topic [SuiteCloud Customization Tutorials](#).

You can also find a complete alphabetized listing of all samples in the SuiteScript Samples Catalog here: [SuiteScript Samples Catalog Complete Listing](#).

SuiteScript Samples by Module

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This section of the SuiteScript Code Samples Catalog includes the module code samples which are provided for individual SuiteScript module APIs. For more information about SuiteScript modules, see the help topic [SuiteScript 2.x Modules](#). For more information about the SuiteScript module APIs, see the help topic [SuiteScript 2.x API Reference](#).

For use case samples available in the SuiteScript Code Samples catalog, see [SuiteScript Use Cases Samples](#). For plug-in samples, see [Custom Plug-in Samples](#).

- [N/action Samples](#)
- [N/auth Samples](#)
- [N/cache Samples](#)
- [N/certificateControl Samples](#)
- [N/commerce/recordView Samples](#)
- [N/compress Samples](#)
- [N/config Samples](#)
- [N/crypto Samples](#)
- [N/crypto/certificate Samples](#)
- [N/currency Samples](#)
- [N/currentRecord Samples](#)
- [N/dataset Samples](#)
- [N/email Samples](#)
- [N/encode Samples](#)
- [N/error Samples](#)
- [N/file Samples](#)
- [N/format Samples](#)
- [N/format/i18n Samples](#)
- [N/http Samples](#)
- [N/https Samples](#)
- [N/https/clientCertificate Samples](#)
- [N/keyControl Samples](#)
- [N/log Samples](#)
- [N/piremoval Samples](#)
- [N/plugin Samples](#)
- [N/portlet Samples](#)
- [N/query Samples](#)
- [N/record Samples](#)
- [N/recordContext Samples](#)
- [N/redirect Samples](#)
- [N/render Samples](#)
- [N/runtime Samples](#)
- [N/search Samples](#)
- [N/sftp Samples](#)

- [N/sso Samples](#)
- [N/suiteAppInfo Samples](#)
- [N/task Samples](#)
- [N/task/accounting/recognition Samples](#)
- [N/transaction Samples](#)
- [N/translation Samples](#)
- [N/ui/dialog Samples](#)
- [N/ui/message Samples](#)
- [N/ui/serverWidget Samples](#)
- [N/url Samples](#)
- [N/util Samples](#)
- [N/workbook Samples](#)
- [N/workflow Samples](#)
- [N/xml Samples](#)

N/action Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/action Module](#):

- [Execute a Bulk Action on a Timebill Record](#)
- [Find Actions Available for the Timebill Record Asynchronously Using Promise Methods](#)
- [Locate and Execute an Action on a Timebill Record](#)

Execute a Bulk Action on a Timebill Record

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to execute a bulk approve action on a timebill record using different parameters.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4  require(['N/action', 'N/util'])function(action, util) {
5
6      // 1a) Bulk execute the specified action on a provided list of record IDs.
7      // The params property is an array of parameter objects where each object contains required recordId and arbitrary additional parameters.
8      var handle = action.executeBulk({
9          recordType: "timebill",
10         id: "approve",
11         params: [{
12             recordId: 1,
13             note: "this is a note for 1"
14         },
15         {
16             recordId: 5,
17             note: "this is a note for 5"

```

```

18         },
19         {
20             recordId: 23,
21             note: "this is a note for 23"
22         }
23     ]
24 })
25 });
26
27 // 1b) Bulk execute the specified action on a provided list of record IDs.
28 // The parameters in the previous sample are similar and can be generated programatically using the map function.
29 var searchResults = /* result of a search, for example, [1, 5, 23] */ ;
30 var handle = action.executeBulk({
31     recordType: "timebill",
32     id: "approve",
33     params: searchResults.map(function(v) {
34         return {
35             recordId: v,
36             note: "this is a note for " + v
37         };
38     })
39 });
40
41 // 2a) Bulk execute the specified action on a provided list of record IDs.
42 // This time with homogenous parameters, that is, all parameter objects are equal except recordId.
43 var handle = action.executeBulk({
44     recordType: "timebill",
45     id: "approve",
46     params: searchResults.map(function(v) {
47         return {
48             recordId: v,
49             foo: "bar",
50             name: "John Doe"
51         };
52     })
53 });
54
55 // 2b) Bulk execute the specified action on a provided list of record IDs.
56 // This time with homogenous parameters. Equivalent to the previous sample.
57 var commonParams = {
58     foo: "bar",
59     name: "John Doe"
60 };
61 var handle = action.executeBulk({
62     recordType: "timebill",
63     id: "approve",
64     params: searchResults.map(function(v) {
65         return util.extend({
66             recordId: v
67         }, commonParams);
68     })
69 });
70
71 // 3) Bulk execute the specified action on a provided list of record IDs.
72 // This is the simplest usage with no extra parameters besides the record ID.
73 var handle = action.executeBulk({
74     recordType: "timebill",
75     id: "approve",
76     params: searchResults.map(function(v) {
77         return {
78             recordId: v
79         }
80     })
81 });
82
83 // 4) Bulk execute the specified action on all record instances that qualify.
84 // Since we don't have a list of recordIds in hand, we only provide the callback
85 // that will later be used to transform a recordId to the corresponding parameters object.
86 var handle = action.executeBulk({
87     recordType: "timebill",
88     id: "approve",
89     condition: action.ALL_QUALIFIED_INSTANCES,
90     paramCallback: function(v) {

```

```

91         return {
92             recordId: v,
93             note: "this is a note for " + v
94         };
95     }
96 });
97
98 // 5) Get a particular action for a particular record type.
99 var approveTimebill = action.get({
100     recordType: "timebill",
101     id: "approve"
102 });
103
104 // 6) Bulk execute the previously obtained action on a provided list of record IDs.
105 // Params are generated the same way as above in action.executeBulk().
106 var handle = approveTimebill.executeBulk({
107     params: searchResults.map(function(v) {
108         return {
109             recordId: v,
110             note: "this is a note for " + v
111         };
112     })
113 });
114
115 // 7) Bulk execute the previously obtained action on all record instances that qualify.
116 var handle = approveTimebill.executeBulk({
117     condition: action.ALL_QUALIFIED_INSTANCES,
118     paramCallback: function(v) {
119         return {
120             recordId: v,
121             note: "this is a note for " + v
122         };
123     }
124 });
125
126 // 8) Get status of a bulk action execution.
127 var res = action.getBulkStatus({
128     taskId: handle
129 }); // returns a RecordActionTaskStatus object
130 log.debug(res.status);
131 });

```

Find Actions Available for the Timebill Record Asynchronously Using Promise Methods

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample asynchronously finds actions available for a timebill record and then executes one with promises.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType ClientScript
4   */
5  require(['N/action', 'N/record'], function(action, record) {
6      // create timebill record
7      var rec = record.create({
8          type: 'timebill',
9          isDynamic: true
10     });

```

```

11     rec.setValue({
12         fieldId: 'employee',
13         value: 104
14     });
15     rec.setValue({
16         fieldId: 'location',
17         value: 312
18     });
19     rec.setValue({
20         fieldId: 'hours',
21         value: 5
22     });
23     var recordId = rec.save();
24
25     // find all qualified actions and then execute approve if available
26     action.find.promise({
27         recordType: 'timebill',
28         recordId: recordId
29     }).then(function(actions) {
30         console.log("We've got the following actions: " + Object.keys(actions));
31         if (actions.approve) {
32             actions.approve.promise().then(function(result) {
33                 console.log("Timebill has been successfully approved");
34             });
35         } else {
36             console.log("The timebill is already approved");
37         }
38     });
39 });
40
41 // Outputs the following:
42 // We've got the following actions:
43 // The timebill has been successfully approved

```

Locate and Execute an Action on a Timebill Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample finds and executes an action on the timebill record without promises.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4  require(['N/action', 'N/record'], function(action, record) {
5      // create timebill record
6      var rec = record.create({
7          type: 'timebill',
8          isDynamic: true
9      });
10     rec.setValue({
11         fieldId: 'employee',
12         value: 104
13     });
14     rec.setValue({
15         fieldId: 'location',
16         value: 312
17     });
18     rec.setValue({
19         fieldId: 'hours',
20         value: 5
21     });

```

```

22     var recordId = rec.save();
23
24     var actions = action.find({
25         recordType: 'timebill',
26         recordId: recordId
27     });
28
29
30     log.debug("We've got the following actions: " + Object.keys(actions));
31     if (actions.approve) {
32         var result = actions.approve();
33         log.debug("Timebill has been successfully approved");
34     } else {
35         log.debug("The timebill is already approved");
36     }
37 });
38
39 // Outputs the following:
40 // We've got the following actions: approve, reject
41 // Timebill has been successfully approved

```

N/auth Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/auth Module](#):

- [Change a NetSuite Email Address and Password](#)

Change a NetSuite Email Address and Password

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to change the currently logged-in user's NetSuite email address and password.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

x Warning: When you run this sample code in the SuiteScript Debugger, it logs a real request to change the email address and then changes the password.

! Important: The values used in this sample for the password and email fields are placeholders. Before using this sample, replace the password and email field values with valid values from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  // This script changes the currently logged-in user's NetSuite email address and password.
6  require(['N/auth'], function(auth) {
7      function changeEmailAndPassword() {
8          var password = 'myCurrentPassword';
9          auth.changeEmail({
10             password: password,

```



```

11     newEmail: 'auth_test@newemail.com'
12   });
13   auth.changePassword({
14     currentPassword: password,
15     newPassword: 'myNewPa55Word'
16   });
17 }
18 changeEmailAndPassword();
19 });

```

N/cache Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/cache Module](#).

- [Look Up Folder IDs](#)
- [Retrieve Name of a City Based on a ZIP Code Using Cache and a Custom Loader Function](#)

Look Up Folder IDs

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a cache to help you lookup folder IDs. Folder lookups often require multiple searches. Using a cache could provide a simpler way to find your folder IDs.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

In this sample, the cache keys are the folder names and the cache values are the folder IDs. This sample looks up folders by their folder name and their parent folder name. This sample includes four functions: `folderCacheLoader`, `getFolderCache`, `folderKey`, and `getFolder`.

You would include this sample code in a custom module that is called using the `options.loader` parameter of the `Cache.get(options)` method.

```

1  const FOLDER_CACHE_NAME = 'folder_cache';
2
3  function folderCacheLoader(context) {
4    const PARENT_FOLDER_ID = 0;
5    const FOLDER_NAME = 1;
6    const folderCacheKey = context.key.split('/');
7    const parentFolderId = folderCacheKey[PARENT_FOLDER_ID];
8    const folderName = folderCacheKey[FOLDER_NAME];
9
10   var folderId = null;
11   search.create ({
12     type: search.Type.FOLDER,
13     columns: ['internalid'],
14     filters: [
15       ['parent', search.Operator.ANYOF, parentFolderId],
16       'AND',
17       ['name', search.Operator.IS, folderName]
18     ]
19   }).run()
20   .each(function(folder) {
21     folderid = folder.id;
22     return false;

```

```

23     });
24
25     if (!folderId) {
26         var folder = record.create({
27             type: record.Type.FOLDER
28         });
29         folder.setValue({
30             fieldId: 'parent',
31             value: parentFolderId
32         });
33         folder.setValue({
34             fieldId: 'name',
35             value: folderName
36         });
37         folderId = folder.save();
38     }
39     return folderId;
40 }
41
42 function getFolderCache() {
43     return cache.getCache({
44         name: FOLDER_CACHE_NAME
45     });
46 }
47
48 function folderKey(folderName, parentFolderId) {
49     return[parentFolderId, folderName].join('/');
50 }
51
52 function getFolder(folderName, parentFolderId) {
53     return getFolderCache().get({
54         key: folderKey(folderName, parentFolderId),
55         loader: folderCacheLoader
56     });
57 }

```

Retrieve Name of a City Based on a ZIP Code Using Cache and a Custom Loader Function

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a Suitelet and a custom module to retrieve the name of a city based on a ZIP code. To speed up processing, the Suitelet uses a cache.

In this sample, the ZIP code is the key used to retrieve city names from the cache. A loader function is called if the city corresponding to the provided ZIP code (key) is not in the cache. This loader function is a custom module that loads a CSV file and uses it to find the requested value. This function is called `zipCodeDatabaseLoader` (included in the second script sample).

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Note: This sample depends on a CSV file that must exist before the script is run. The sample CSV file is available [here](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

2  * @NApiVersion 2.1
3  * @NScriptType Suitelet
4  */
5
6  // This script retrieves the name of a city based on a ZIP code from a cache.
7  define(['N/cache', '/SuiteScripts/zipToCityIndexCacheLoader'], function(cache, lib) {
8      const ZIP_CODES_CACHE_NAME = 'ZIP_CODES_CACHE';
9      const ZIP_TO_CITY_IDX_JSON = 'ZIP_TO_CITY_IDX_JSON';
10
11     function getZipCodeToCityLookupObj() {
12         const zipCache = cache.getCache({
13             name: ZIP_CODES_CACHE_NAME
14         });
15         const zipCacheJson = zipCache.get({
16             key: ZIP_TO_CITY_IDX_JSON,
17             loader: lib.zipCodeDatabaseLoader
18         });
19         return JSON.parse(zipCacheJson);
20     }
21
22     function findCityByZipCode(options) {
23         return getZipCodeToCityLookupObj()[String(options.zip)];
24     }
25
26     function onRequest(context) {
27         const start = new Date();
28         if (context.request.parameters.purgeZipCache === 'true') {
29             const zipCache = cache.getCache({
30                 name: ZIP_CODES_CACHE_NAME
31             });
32             zipCache.remove({
33                 key: ZIP_TO_CITY_IDX_JSON
34             });
35         }
36         const cityName = findCityByZipCode({
37             zip: context.request.parameters.zipcode
38         });
39
40         context.response.writeln(cityName || 'Unknown :(');
41
42         if (context.request.parameters.auditPerf === 'true') {
43             context.response.writeln('Time Elapsed: ' + (new Date().getTime() - start.getTime()) + ' ms');
44         }
45     }
46     return {
47         onRequest: onRequest
48     };
49 });

```

The following custom module provides the loader function used in the preceding Suitelet script sample. The loader function uses a CSV file to retrieve a value that was missing from a cache. This custom module does not need to include logic for placing the retrieved value into the cache. Whenever a value is returned through the options.loader parameter of the Cache.get(options) method, the value is automatically placed into the cache. This allows the loader function to serve as the sole method of populating a cache with values.

```

1  /**
2   * zipToCityIndexCacheLoader.js
3   * @NApiVersion 2.1
4   * @NModuleScope Public
5   */
6
7  // This custom module is a loader function that uses a CSV file to retrieve a value that was missing from a cache.
8  define(['N/file', 'N/cache'], function(file, cache) {
9      const ZIP_CODES_CSV_PATH = '/SuiteScripts/Resources/free-zipcode-CA-database-primary.csv';
10
11     function trimOuterQuotes(str) {
12         return (str || '').replace(/^"+/, '').replace(/"+$/, '');
13     }
14

```

```

15 function zipCodeDatabaseLoader(context) {
16     log.audit({
17         title: 'Loading Zip Codes',
18         details: 'Loading Zip Codes for ZIP_CODES_CACHE'
19     });
20     const zipCodesCsvText = file.load({
21         id: ZIP_CODES_CSV_PATH
22     }).getContents();
23     const zipToCityIndex = {};
24     const csvLines = zipCodesCsvText.split('\n');
25     util.each(csvLines.slice(1), function(el) {
26         var cells = el.split(',');
27         var key = trimOuterQuotes(cells[0]);
28         var value = trimOuterQuotes(cells[2]);
29         if (parseInt(key, 10))
30             zipToCityIndex[String(key)] = value;
31     });
32     return zipToCityIndex;
33 }
34
35 return {
36     zipCodeDatabaseLoader: zipCodeDatabaseLoader
37 }
38 });

```

N/certificateControl Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/certificateControl Module](#):

- [Create, Modify, and Save Certificate Record Based on a File in the File Cabinet](#)
- [Establish an SFTP Connection Using an SSH Key; Create, Update, Load, and Delete a Certificate Record](#)
- [Filter the Digital Certificate List by Subsidiary and File Type](#)
- [Find and Use an Existing Certificate Record](#)
- [Find the Audit Trail of POST Operations for a Certificate Record Based on ID](#)
- [Generate Signature of a Plain Text String and Verify the Signature Using the Same Certificate](#)

Create, Modify, and Save Certificate Record Based on a File in the File Cabinet

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a file object by loading a file from the File Cabinet. It then creates the options needed for the [certificateControl.createCertificate\(options\)](#) method and creates and saves the certificate record. The certificate record is then loaded again, edited to change the file, and saved again.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2  * @NApiVersion 2.x

```

```

3  */
4
5  require(['N/certificateControl', 'N/file'], function(cc, file){
6      var fileObj = file.load({
7          id: 'SuiteScripts/dsa.p12'
8      });
9      var options = {
10         file : fileObj,
11         password : '022b490ad4334c7e86a8304f937ec68f',
12         name : 'testCert',
13         description : 'testDescription',
14         scriptId : '_testid',
15         subsidiaries : [1,3],
16         weekReminder : false,
17         monthReminder : true,
18         threeMonthsReminder : false
19     };
20     var newCertificate = cc.createCertificate(options);
21     newCertificate.save();
22
23     var loadedCertificate = cc.loadCertificate({
24         scriptId : 'custcertificate_testid'
25     });
26     fileObj = file.load({
27         id: 'SuiteScripts/ecdsa.p12'
28     });
29     loadedCertificate.file = fileObj;
30     loadedCertificate.password = '022b490ad4334c7e86a8304f937ec68f';
31     loadedCertificate.save();
32 })

```

Establish an SFTP Connection Using an SSH Key; Create, Update, Load, and Delete a Certificate Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample establishes a SFTP connection using an SSH key that has already been uploaded to NetSuite. It then creates, updates, loads, and deletes a certificate record to show the full CRUD operation. Replace the server URL with your correct URL.

For the SFTP connection, the public key corresponding to the private key in the certificate must be stored in the .ssh/authorized_keys file on the server.

i Note: This sample script uses the require function so that you can copy it into the SuiteScript Debugger and test it. You must use the define function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/file', 'N/sftp', 'N/certificateControl'], function(file, sftp, certificateControl) {
6      var certPath = 'yyy/certificates'
7      var certName = 'apiclient_cert.p12';
8
9      // Establish SFTP connection
10     log.debug('Establishing secured SFTP connection...');
11     var connection = sftp.createConnection({
12         username: 'sftpuser',
13         keyId: 'custkeysftp_nft_demo_key',
14         url: 'my.sftp.example.com',
15         port: 22,
16         directory: 'inbound',

```

```

17     hostKey: 'AAAA3NzaC1yc2EAAAADAQABAAQAC4gYD1K4IE9QnuYEgRRQChjrAM1+bTT95e71Xv0aQ60ywVQEiedhRqSMbP1CPPB4pjpBdOmPIQC
    Ckug+3XwAQ6uNj3UM11zoG6mg86tyEJT6qGB0SsrQJzHTb3EG3885rB00WEz0WeJ8E8YODT3oAj1NrF8Ls3JbG0bRF+0uwJD11SrFkYS3kWCv27NhBnaytGe7iLBgrJd
    NVlitNqkx2fK0NsAYCaJWQjQLtz+GFfN5zTbKKNsDa6s/YW7oAMMOI3Q5GQAqdXtKY728WvxYTjr2FsYS/KM6nbq/csTvZHWLE0z2TQtB2H0IOfvEP/QvXwmGEnCeVpCn
    gRwdHWQF'
18   });
19   log.debug('Connection established!');
20   // -----
21
22   // Create new certificate
23   log.debug('Creating certificate...');
24   var certScriptId = '_' + (Math.random().toString(36).substring(2, 10));
25   var cert = certificateControl.createCertificate();
26   cert.name = 'Test Certificate China API';
27   cert.description = 'Test Certificate China created using API';
28   // custcertificate prefix will be added automatically
29   cert.scriptId = certScriptId;
30   log.debug('Downloading certificate file from SFTP...');
31   cert.file = connection.download({
32     directory: certPath,
33     filename: certName
34   });
35   log.debug('Successfully downloaded!');
36   //guid corresponding to the certificate's password;
37   var pwd = '022b490ad4334c7e86a8304f937ec68f',
38   cert.password = pwd;
39   cert.save();
40   /**/certScriptId = 'custcertificate' + certScriptId;
41   log.debug('Certificate "' + cert.name + '" successfully created with id "' + certScriptId + '"');
42   // -----
43   // Rename certificate
44   log.debug('Renaming certificate...');
45   cert = certificateControl.loadCertificate({scriptId: certScriptId});
46   cert.name = 'Test Certificate China API TEMP';
47   cert.save();
48   // Verify new certificate name
49   /**/cert = certificateControl.loadCertificate({scriptId: certScriptId});
50   log.debug('Certificate successfully renamed to "' + cert.name + '"');
51   // -----
52   // Delete certificate
53   log.debug('Deleting certificate "' + cert.name + '...');
54   certificateControl.deleteCertificate(certScriptId);
55   log.debug('Certificate deleted!');
56   // -----
57   // Load the deleted certificate
58   log.debug('Attempting to load the deleted certificate to invoke an error...');
59   try {
60     cert = certificateControl.loadCertificate({scriptId: certScriptId});
61   }
62   catch (e) {
63     log.error(e.message);
64   }
65 })

```

Filter the Digital Certificate List by Subsidiary and File Type

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to filter the Digital Certificates list by subsidiary and by file type.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and depoly). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2  * @NApiVersion 2.x

```

```

3  */
4  require(['N/certificateControl'],
5      function(certificateControl){
6          var all = certificateControl.findCertificates();
7          var specificType = certificateControl.findCertificates({
8              type: 'PFX'
9          });
10         var specificSub = certificateControl.findCertificates({
11             subsidiary: 93
12         });
13         var specificTypeAndSub = certificateControl.findCertificates({
14             type: 'PFX',
15             subsidiary: 93
16         });

```

Find and Use an Existing Certificate Record

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to find an existing certificate record and use it in an operation.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/certificateControl','N/https/clientCertificate'],function(cc, cert){
6      var yodlee = cc.findCertificates({
7          name: 'Yodlee',
8          description: 'Yodlee certificate'
9      });
10     cert.post({certId:yodlee[0].id,url:<url>, body:<body>, headers:<headers>
11     });
12 })

```

Find the Audit Trail of POST Operations for a Certificate Record Based on ID

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to find the audit trail of POST operations for the certificate record with ID 'custcertificate_china'.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/certificateControl'], function(cc){
6      var usages = cc.findUsages({

```

```

7         id: 'custcertificate_china',
8         operation: cc.Operation.POST
9     });
10 })

```

Generate Signature of a Plain Text String and Verify the Signature Using the Same Certificate

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to generate a signature of a plaintext string and then verifies the signature using the same certificate.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/certificateControl', 'N/crypto/certificate'], function(cc, certificate){
6      var signer = certificate.createSigner({
7          certId: 'custcertificate_cert_1',
8          algorithm: 'SHA256'
9      });
10     var result = signer.sign();
11     var verifier = certificate.createVerifier({
12         certId: 'custcertificate_cert_1',
13         algorithm: 'SHA256'
14     });
15     verifier.update('test');
16     verifier.verify(result);
17     var res = cc.findUsages();
18     ;
19 })

```

The `res` variable returns an array of information about the usage of the digital certificate, including the date of the action, the type of operation, such as `sign`, and the internal ID of the person who performed the action.

N/commerce/recordView Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/commerce/recordView Module](#):

Retrieve Website and Item Data

Retrieve Website and Item Data

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample retrieves some details of the website and some item data for the specified items.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4   define(['N/commerce/recordView'],
5     function (recordView) {
6       function service(context) {
7         var result = {};
8         try {
9           result.website= recordView.viewWebsite({
10             id: 2,
11             fields: ["internalid","shiptocountries"]
12           });
13         }
14         catch (e) {
15           result.websiteError = e.name + ": " + e.message;
16         }
17
18         var options = {
19           "ids": [382,388],
20           "fields": ["displayname"]
21         };
22         try {
23           result.items= recordView.viewItems(options);
24         }
25         catch (e) {
26           result.itemsError = e.name + ": " + e.message;
27         }
28         return context.response.write(
29           JSON.stringify(result)
30         );
31       }
32
33       return {
34         service: service
35       };
36     }
37 );

```

N/compress Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/compress Module](#):

- [Compress and Decompress a File](#)
- [Create a ZIP File](#)

Compress and Decompress a File

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample compresses and decompresses a file.

Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topic [SuiteScript 2.x Script Basics](#) SuiteScript 2.0 Script Basics and [SuiteScript 2.x Script Types](#) SuiteScript 2.0 Script Types.

```

1 require(['N/compress', 'N/file'], function(compress, file) {
2   var unsavedTxtFile = file.create({
3     fileType: 'PLAINTEXT',
4     name: 'file.txt',
5     contents: 'This is a sample content. This is a sample content. This is a sample content. This is only a sample.'
6   });
7
8   log.debug('#### ORIGINAL FILE #### #');
9   log.debug('Name: ' + unsavedTxtFile.name);
10  log.debug('Size: ' + unsavedTxtFile.size + 'b');
11  log.debug('Contents: ' + unsavedTxtFile.getContents());
12
13  log.debug('#### GZIPPED FILE WITH MAX COMPRESSION ####');
14  var gzippedFile = compress.zip({
15    file: unsavedTxtFile,
16    level: 9
17  });
18  log.debug('Name: ' + gzippedFile.name);
19  log.debug('Size: ' + gzippedFile.size + 'b');
20  log.debug('Contents: ' + gzippedFile.getContents().substring(0, 100));
21
22  log.debug('#### GUNZIPPED FILE ####');
23  var gunzippedFile = compress.gunzip({
24    file: gzippedFile
25  });
26  log.debug('Name: ' + gunzippedFile.name);
27  log.debug('File size: ' + gunzippedFile.size + 'b');
28  log.debug('Contents: ' + gunzippedFile.getContents());
29
30  });
31  });

```

Create a ZIP File

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a ZIP file.

Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topic [SuiteScript 2.x Script Basics](#) SuiteScript 2.0 Script Basics and [SuiteScript 2.x Script Types](#) SuiteScript 2.0 Script Types.

```

1 require(['N/compress', 'N/file'], function(compress, file) {
2   // load/create files to be archived
3   var binaryFile = file.load({
4     id: 200
5   });
6   var textFile = file.create({
7     name: 'file.txt',
8     fileType: 'PLAINTEXT',
9     contents: 'This is sample content.'
10  });
11
12  // create an archive as a temporary file object
13  var archiver = compress.createArchiver();
14  archiver.add({
15    file: binaryFile

```

```

16     });
17     archiver.add({
18         file: textFile,
19         directory: 'txt/'
20     });
21     var zipFile = archiver.archive({
22         name: 'myarchive.zip'
23     });
24
25     // save the archive to file cabinet
26     zipFile.folder = 123;
27     zipFile.save();
28 });

```

N/config Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/config Module](#):

- [Load the Company Information Configuration Page and Set Field Values](#)

Load the Company Information Configuration Page and Set Field Values

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads the Company Information configuration page. It then sets the values specified for the Tax ID Number field and the Employer Identification Number field.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

i Note: The IDs in this sample are placeholders. Replace the values of the Tax ID Number field and the Employer Identification Number field with valid IDs from your NetSuite account.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/config'],
6      function(config) {
7          function setTaxAndEmployerId() {
8              var companyInfo = config.load({
9                  type: config.Type.COMPANY_INFORMATION
10             });
11             companyInfo.setValue({
12                 fieldId: 'taxid',
13                 value: '1122334455'
14             });
15             companyInfo.setValue({
16                 fieldId: 'employerid',
17                 value: '123456789'
18             });
19             companyInfo.save();
20             companyInfo = config.load({
21                 type: config.Type.COMPANY_INFORMATION

```

```

22     });
23     var taxid = companyInfo.getValue({
24         fieldId: 'taxid'
25     });
26 }
27 setTaxAndEmployerId();
28 });

```

N/crypto Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/crypto Module](#):

- [Create a Secure Key Using SHA512](#)
- [Create a Suitelet to Request User Credentials, Create a Secret Key, and Encode a Sample String](#)

Create a Secure Key Using SHA512

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample demonstrates the APIs needed to generate a secure key using the SHA512 hashing algorithm. The GUID in this sample is a placeholder. You must replace it with a valid value from your NetSuite account. To create a real password GUID, obtain a password value from a credential field on a form. For more information, see the help topic [Form.addCredentialField\(options\)](#). Also see the [Create a Suitelet to Request User Credentials, Create a Secret Key, and Encode a Sample String](#) that shows how to create a form field that generates a GUID.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  /**This sample demonstrates the APIs needed to generate a secure key using the SHA512 hashing algorithm.
5   * The mySecret variable is a placeholder that must be replaced with a valid secret from your NetSuite account.
6   * For information about secrets management page, see Account Administration > Authentication > Secrets Management
7   * in the Help Center.
8   */
9
10 require(['N/crypto', 'N/encode', 'N/runtime'], (crypto, encode, runtime) => {
11     function createSecureKeyWithHash() {
12         let mySecret = 'custsecret_my_secret'; //secret id take from secrets management page
13
14         let sKey = crypto.createSecretKey({
15             secret: mySecret,
16             encoding: encode.Encoding.UTF_8
17         });
18
19         let hmacSHA512 = crypto.createHmac({
20             algorithm: crypto.HashAlg.SHA512,
21             key: sKey
22         });
23
24         hmacSHA512.update({
25             input: inputString,
26             inputEncoding: encode.Encoding.BASE_64
27         });

```

```

28
29     let digestSHA512 = hmacSHA512.digest({
30         outputEncoding: encode.Encoding.HEX
31     });
32 }
33 createSecureKeyWithHash();
34 });

```

Create a Suitelet to Request User Credentials, Create a Secret Key, and Encode a Sample String

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a simple Suitelet that requests user credentials, creates a secret key, and encodes a sample string.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

i Note: The default maximum length for a secret key field is 32 characters. If needed, use the `Field.maxLength` property to change this value.

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/ui/serverWidget', 'N/runtime', 'N/crypto', 'N/encode'], (ui, runtime, crypto, encode) => {
6      function onRequest(option) {
7          if (option.request.method === 'GET') {
8              let form = ui.createForm({
9                  title: 'My Credential Form'
10             });
11             let skField = form.addSecretKeyField({
12                 id: 'mycredential',
13                 label: 'Credential',
14                 restrictToScriptIds: [runtime.getCurrentScript().id],
15                 restrictToCurrentUser: false
16             });
17             skField.maxLength = 200;
18
19             form.addSubmitButton();
20
21             option.response.writePage(form);
22         } else {
23             let form = ui.createForm({
24                 title: 'My Credential Form'
25             });
26
27             const inputString = "YWJjZGVmZwo=";
28             let myGuid = option.request.parameters.mycredential;
29
30             // Create the key
31             let sKey = crypto.createSecretKey({
32                 guid: myGuid,
33                 encoding: encode.Encoding.UTF_8
34             });
35
36             try {
37                 let hmacSha512 = crypto.createHmac({
38                     algorithm: 'SHA512',
39                     key: sKey

```

```

40     });
41     hmacSha512.update({
42         input: inputString,
43         inputEncoding: encode.Encoding.BASE_64
44     });
45     let digestSha512 = hmacSha512.digest({
46         outputEncoding: encode.Encoding.HEX
47     });
48     } catch (e) {
49         log.error({
50             title: 'Failed to hash input',
51             details: e
52         });
53     }
54
55     form.addField({
56         id: 'result',
57         label: 'Your digested hash value',
58         type: 'textarea'
59     }).defaultValue = digestSha512;
60
61     option.response.writePage(form);
62 }
63 }
64 return {
65     onRequest: onRequest
66 };
67 });

```

N/crypto/certificate Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/crypto/certificate Module](#):

- [Create Signer and Verifier Objects](#)
- [Load an XML File from the File Cabinet and Sign It Using a Digital Certificate](#)

Create Signer and Verifier Objects

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a [certificate.Signer](#) object, signs it, and then creates a [certificate.Verifier](#) object and verifies the signer object.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5   require(['N/crypto/certificate'], (certificate) => {
6       let signer = certificate.createSigner({
7           certId: 'custcertificate1',

```

```

8      algorithm: certificate.HashAlg.SHA256
9    });
10    signer.update('test');
11
12    let result = signer.sign();
13    let verifier = certificate.createVerifier({
14      certId: 'custcertificate1',
15      algorithm: certificate.HashAlg.SHA256
16    });
17    verifier.update('test');
18    verifier.verify(result);
19  })

```

Load an XML File from the File Cabinet and Sign It Using a Digital Certificate

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads an XML file from the File Cabinet and signs it using the digital certificate with internal ID 'custcertificate1'. Note that this sample uses a hard-coded value for the file id. You should change this value to a valid file id from your account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5  require(['N/crypto/certificate', 'N/file'], (cert, file) => {
6    // Load the file from the File Cabinet.
7    // Note that the id value is hard-coded in this sample, and you should use
8    // a valid file id from your account.
9    let infNFe = file.load({
10     id: 922
11   });
12   let signedXml = cert.signXml({
13     algorithm: certificate.HashAlg.SHA256,
14     certId: 'custcertificate1',
15     rootTag: 'infNFe',
16     xmlString: infNFe.getContents()
17   });
18   cert.verifyXMLSignature({
19     signedXml: signedXml,
20     rootTag: 'infNFe'
21   });
22   }); // Note that this value is hard-coded in this sample, and you should use

```

N/currency Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/currency Module](#):

- [Obtain the Exchange Rate Between the Canadian Dollar and the U.S. Dollar](#)

Obtain the Exchange Rate Between the Canadian Dollar and the U.S. Dollar

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to obtain the exchange rate between the Canadian dollar and the U.S. dollar on a specific date.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/currency'], function(currency) {
6      function getUSDFromCAD() {
7          var canadianAmount = 100;
8          var rate = currency.exchangeRate({
9              source: 'CAD',
10             target: 'USD',
11             date: new Date('7/28/2015')
12         });
13
14         var usdAmount = canadianAmount * rate;
15     }
16
17     getUSDFromCAD();
18 });

```

N/currentRecord Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/currentRecord Module](#):

- [Perform Field Sourcing Synchronously](#)
- [Update Fields on Current Record](#)
- [Use a Custom Module Client Script](#)

Perform Field Sourcing Synchronously

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following script sample shows how to use the `forceSyncSourcing` parameter.

This parameter can be used to alleviate a timing situation that may occur in some browsers when fields are sourced. For some browsers, some APIs are triggered without waiting for the field sourcing to

complete. For example, if `forceSyncSourcing` is set to `false` when adding sublist lines, the lines aren't committed as expected. Setting the parameter to `true`, forces synchronous sourcing.

This sample shows using the `forceSyncSourcing` parameter in the `setCurrentSublistValue` method. The `forceSyncSourcing` parameter is also available in the `setText`, `setValue`, `setCurrentSublistText`, `setMatrixHeaderValue`, and `setCurrentMatrixSublistValue` methods.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/currentRecord'], function(currentRecord){
6      var rec = currentRecord.get();
7      rec.selectNewLine({
8          sublistId: 'item'
9      });
10     rec.setCurrentSublistValue({
11         sublistId: 'item',
12         fieldId: 'item',
13         value: 39,
14         forceSyncSourcing:true
15     });
16     rec.setCurrentSublistValue({
17         sublistId: 'item',
18         fieldId: 'quantity',
19         value: 1,
20         forceSyncSourcing:true
21     });
22     rec.commitLine({sublistId: 'item'});
23 })

```

Update Fields on Current Record

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample is a custom module client script named `clientDemo.js`. This script updates fields on the current record. After you upload the `clientDemo.js` script file to a NetSuite account, it can be called by other scripts. The sample script that follows this one (Sample 2) shows how to call the methods defined in this client script.

Because `clientDemo.js` is a custom module script, it must manually load the [N/currentRecord Samples](#) method by naming it in the `define` statement. It must also retrieve a `currentRecord.CurrentRecord` object by using the `currentRecord.get()` method.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.1

```

```

3  */
4  define(['N/currentRecord'], currentRecord => {
5      return ({
6          test_set_getValue: () => {
7              // Get a reference to the currently active record
8              let myRecord = currentRecord.get();
9
10             // Set the value of a custom field
11             myRecord.setValue({
12                 fieldId: 'custpage_textfield',
13                 value: 'Body value',
14                 ignoreFieldChange: true,
15                 forceSyncSourcing: true
16             });
17
18             // Retrieve the value that was set
19             let actValue = myRecord.getValue({
20                 fieldId: 'custpage_textfield'
21             });
22
23             // Set the value of another custom field
24             myRecord.setValue({
25                 fieldId: 'custpage_resultfield',
26                 value: actValue,
27                 ignoreFieldChange: true,
28                 forceSyncSourcing: true
29             });
30         },
31
32         test_set_getCurrentSublistValue: () => {
33             // Get a reference to the currently active record
34             let myRecord = currentRecord.get();
35
36             // Set the value of a custom sublist field
37             myRecord.setCurrentSublistValue({
38                 sublistId: 'sitecategory',
39                 fieldId: 'custpage_subtextfield',
40                 value: 'Sublist Value',
41                 ignoreFieldChange: true,
42                 forceSyncSourcing: true
43             });
44
45             // Retrieve the value that was set
46             let actValue2 = myRecord.getCurrentSublistValue({
47                 sublistId: 'sitecategory',
48                 fieldId: 'custpage_subtextfield'
49             });
50
51             // Set the value of another custom field
52             myRecord.setValue({
53                 fieldId: 'custpage_sublist_resultfield',
54                 value: actValue2,
55                 ignoreFieldChange: true,
56                 forceSyncSourcing: true
57             });
58         }
59     });
60 });

```

Use a Custom Module Client Script

📌 Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample is a user event script deployed on a non-inventory item record. Before the record loads, the script updates the form used by the record to add new text fields, a sublist, and buttons that call the clientDemo.js methods. The buttons access the current record and set values for some of the form's fields. This sample demonstrates how to customize a form, use the code you created in Sample 1, and see the new fields and buttons in action.

This sample depends on the sample script that precedes it (Sample 1), and you should use both of these scripts together in your account.



Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6  define([], () => {
7      return {
8          beforeLoad: params => {
9              // Get a reference to the current form that is about to load
10             let form = params.form;
11
12             // Add several custom fields to the form
13             let textfield = form.addField({
14                 id: 'custpage_textfield',
15                 type: 'text',
16                 label: 'Text'
17             });
18             let resultfield = form.addField({
19                 id: 'custpage_resultfield',
20                 type: 'text',
21                 label: 'Result'
22             });
23             let sublistResultfield = form.addField({
24                 id: 'custpage_sublist_resultfield',
25                 type: 'text',
26                 label: 'Sublist Result Field'
27             });
28
29             // Get a reference to the sitecategory sublist
30             let sublistObj = form.getSublist({
31                 id: 'sitecategory'
32             });
33
34             // Add a custom field to the sublist
35             let subtextfield = sublistObj.addField({
36                 id: 'custpage_subtextfield',
37                 type: 'text',
38                 label: 'Sublist Text Field'
39             });
40
41             // Set the module path to the previous sample script
42             form.clientScriptModulePath = './clientDemo.js';
43
44             // Add two custom buttons to the form
45             form.addButton({
46                 id: 'custpage_custombutton',
47                 label: 'SET_GET_VALUE',
48                 functionName: 'test_set_getValue'
49             });
50             form.addButton({
51                 id: 'custpage_custombutton2',
52                 label: 'SET_GETCURRENTSUBLISTVALUE',
53                 functionName: 'test_set_getCurrentSublistValue'
54             });
55         }
56     };
57 });

```

N/dataset Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/dataset Module](#):

- [Create a Dataset, Run the Dataset, and List All Existing Datasets](#)
- [List All Datasets and Load the First Dataset](#)

Create a Dataset, Run the Dataset, and List All Existing Datasets

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a dataset with three columns, one of which is based on a join. Then, all datasets are listed and the newly created dataset is loaded and reviewed.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  // The following script creates a dataset with three columns, one of which is based on a join, and a condition. Then, all datasets are listed and the newly
   // created dataset is loaded and reviewed.
6
7  require(['N/dataset', 'N/query'], function(dataset, query){
8      var myTransactionDateColumn = dataset.createColumn({
9          fieldId: 'trandate',
10         alias: 'date'
11     });
12     var myJoin = dataset.createJoin({
13         fieldId: 'createdby',
14         target: 'entity'
15     });
16     var myNameColumn = dataset.createColumn({
17         fieldId: 'lastname',
18         alias: 'name',
19         join: myJoin
20     });
21     var myTransactionIdColumn = dataset.createColumn({
22         fieldId: 'tranid',
23         alias: 'id'
24     });
25     var myTotalColumn = dataset.createColumn({
26         fieldId: 'foreigntotal',
27         alias: 'total'
28     });
29     var myColumns = [myTransactionIdColumn, myNameColumn, myTransactionDateColumn, myTotalColumn];
30     var myCondition = dataset.createCondition({
31         column: myNameColumn,
32         operator: query.Operator.EMPTY
33     });
34     var myDataset = dataset.create({

```

```

35     type: 'transaction',
36     columns: myColumns,
37     condition: myCondition
38   });
39
40   // List all created datasets
41   var allDatasets = dataset.list();
42   log.debug({
43     title: 'All datasets:',
44     details: allDatasets
45   });
46
47   // Review the newly created dataset components (in the log)
48   log.debug({
49     title: 'My Dataset = ',
50     details: myDataset
51   });
52
53   // Now run the dataset
54   var runResult = myDataset.run();
55   var runPagedResult = myDataset.runPaged({
56     pageSize: 2
57   });
58   log.debug({
59     title: 'runResult: ',
60     details: runResult
61   });
62   log.debug({
63     title: 'runPagedResult: ',
64     details: runPagedResult
65   });
66 });

```

List All Datasets and Load the First Dataset

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample lists all existing datasets and then loads the first dataset.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // The following script lists all existing datasets and then loads the first dataset.
6
7   require(['N/dataset'], function(dataset){
8     // List all created datasets
9     var allDatasets = dataset.list();
10    log.debug({
11      title: 'All datasets:',
12      details: allDatasets
13    });
14
15    // Load the first dataset
16    var myFirstDataset = dataset.load({
17      id: allDatasets[0].id
18    });
19    log.debug('myFirstDataset:', myFirstDataset);
20  });

```

N/email Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/email Module](#):

■ Send an Email with an Attachment

Send an Email with an Attachment

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to send an email with an attachment.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

i Note: Some of the values in this sample are placeholders, such as the `senderId` and `recipientEmail` values. Before using this sample, replace all hard-coded values, including IDs and file paths, with valid values from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

⚠ Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5   // This script sends an email with an attachment.
6
7   require(['N/email', 'N/record', 'N/file'], (email, record, file) => {
8     const senderId = -515;
9     const recipientEmail = 'notify@myCompany.com';
10    let timeStamp = new Date().getUTCMilliseconds();
11
12    let recipient = record.create({
13      type: record.Type.CUSTOMER,
14      isDynamic: true
15    });
16    recipient.setValue({
17      fieldId: 'subsidiary',
18      value: '1'
19    });
20    recipient.setValue({
21      fieldId: 'companyname',
22      value: 'Test Company' + timeStamp
23    });
24    recipient.setValue({
25      fieldId: 'email',
26      value: recipientEmail
27    });
28
29    let recipientId = recipient.save();
30
31    let fileObj = file.load({

```

```

32     id: 88
33   });
34
35   email.send({
36     author: senderId,
37     recipients: recipientId,
38     subject: 'Test Sample Email Module',
39     body: 'email body',
40     attachments: [fileObj],
41     relatedRecords: {
42       entityId: recipientId,
43       customRecord: {
44         id: recordId,
45         recordType: recordTypeId
46       }
47     }
48   });
49 });

```

N/encode Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/encode Module](#):

■ Convert a String to a Different Encoding

Convert a String to a Different Encoding

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following script shows how to convert a string to a different encoding.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/encode'], function(encode) {
6    function convertStringToDifferentEncoding() {
7      var stringInput = "TÄfÄst StriÄg Input";
8      var base64EncodedString = encode.convert({
9        string: stringInput,
10       inputEncoding: encode.Encoding.UTF_8,
11       outputEncoding: encode.Encoding.BASE_64
12     });
13     var hexEncodedString = encode.convert({
14       string: stringInput,
15       inputEncoding: encode.Encoding.UTF_8,
16       outputEncoding: encode.Encoding.HEX
17     });
18   }
19
20   convertStringToDifferentEncoding();
21 });

```

N/error Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/error Module](#):

- [Create a Custom Error](#)
- [Create an Error Based on a Condition](#)

Create a Custom Error

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a custom error.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script creates a custom error.
6   require(['N/error'], function(error) {
7     function createError() {
8       var myCustomError = error.create({
9         name: 'MY_ERROR_CODE',
10        message: 'My custom error details',
11        notifyOff: true
12      });
13    }
14    createError();
15  });

```

Create an Error Based on a Condition

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to conditionally create and throw a custom error.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script conditionally creates and throws an error.
6   require(['N/error'], function(error) {
7     function showError() {
8       var someVariable = false;
9     }

```



```

10     if (!someVariable) {
11         var myCustomError = error.create({
12             name: 'WRONG_PARAMETER_TYPE',
13             message: 'Wrong parameter type selected.',
14             notifyOff: false
15         });
16
17         // This will write 'Error: WRONG_PARAMETER_TYPE Wrong parameter type selected' to the log
18         log.error('Error: ' + myCustomError.name, myCustomError.message);
19         throw myCustomError;
20     }
21 }
22 showError();
23 });

```

N/file Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/file Module](#):

- [Create a File, Set Property Values, and Save It to the File Cabinet](#)
- [Create and Save a CSV File then Reload the File and Parse Its Contents](#)
- [Create and Save a File to the File Cabinet](#)
- [Read and Log File Contents Using Commas and New Lines as Separators](#)
- [Read and Log Segments of a File Using a Set of Characters as Separators](#)

Create a File, Set Property Values, and Save It to the File Cabinet

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create and save a file to the File Cabinet. It also shows how to set the values of the [File.isOnline](#) and the [File.folder](#) properties. In this sample, the folder ID value is hard-coded. For the script to run in the SuiteScript Debugger, you must replace this hard-coded value with a valid folder ID from your account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/file'], file => {
5       // Create a file containing text
6       // Note that the folder value is hard-coded in this sample, and you should
7       // use a valid folder ID from your account
8       let fileObj = file.create({
9           name: 'testHelloWorld3.txt',
10          fileType: file.Type.PLAINTEXT,

```

```

11     contents: 'Hello World\nHello World',
12     folder: -15,
13     isOnline: true
14 });
15
16 // Save the file
17 let id = fileObj.save();
18
19 // Load the same file to ensure it was saved correctly
20 fileObj = file.load({
21     id: id
22 });
23 });

```

Create and Save a CSV File then Reload the File and Parse Its Contents

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a CSV file, appends several lines of data, and saves the file. The script also loads the file and calculates the total of several values in the file.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/file', 'N/error', 'N/log'], function (file, error, log) {
6      // This sample calculates the total for the
7      // second column value in a CSV file.
8      //
9      // Each line in the CSV file has the following format:
10     // date,amount
11     //
12     // Here is the data that the script adds to the file:
13     // 10/21/14,200.0
14     // 10/21/15,210.2
15     // 10/21/16,250.3
16
17     // Create the CSV file
18     var csvFile = file.create({
19         name: 'data.csv',
20         contents: 'date,amount\n',
21         folder: 39,
22         fileType: 'CSV'
23     });
24
25     // Add the data
26     csvFile.appendLine({
27         value: '10/21/14,200.0'
28     });
29     csvFile.appendLine({
30         value: '10/21/15,210.2'
31     });
32     csvFile.appendLine({
33         value: '10/21/16,250.3'
34     });
35
36     // Save the file
37     var csvFileId = csvFile.save();

```

```

38
39 // Create a variable to store the calculated total
40 var total = 0.0;
41
42 // Load the file
43 var invoiceFile = file.load({
44     id: csvFileId
45 });
46
47 // Obtain an iterator to process each line in the file
48 var iterator = invoiceFile.lines.iterator();
49
50 // Skip the first line, which is the CSV header line
51 iterator.each(function () {return false;});
52
53 // Process each line in the file
54 iterator.each(function (line) {
55     // Update the total based on the line value
56     var lineValues = line.value.split(',');
57     var lineAmount = parseFloat(lineValues[1]);
58     if (!lineAmount) {
59         throw error.create({
60             name: 'INVALID_INVOICE_FILE',
61             message: 'Invoice file contained non-numeric value for total: ' + lineValues[1]
62         });
63
64         total += lineAmount;
65         return true;
66     }
67 });
68
69 // At this point, the total is 660.5
70 log.debug({
71     title: 'total',
72     details: total
73 });
74 });

```

Create and Save a File to the File Cabinet

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create and save a file to the File Cabinet. In this sample, the folder ID value is hard-coded. For the script to run in the SuiteScript Debugger, you must replace this hard-coded value with a valid folder ID from your account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/file'], file => {
5       // Create a file containing text
6       let fileObj = file.create({
7           name: 'testHelloWorld.txt',
8           fileType: file.Type.PLAINTEXT,
9           contents: 'Hello World\nHello World'
10      });

```

```

11 // Set the folder for the file
12 // Note that this value is hard-coded in this sample, and you should use
13 // a valid folder ID from your account
14 fileObj.folder = -15;
15
16
17 // Save the file
18 let id = fileObj.save();
19
20 // Load the same file to ensure it was saved correctly
21 fileObj = file.load({
22   id: id
23 });
24 });

```

Read and Log File Contents Using Commas and New Lines as Separators

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample reads and logs strings from a file using commas and new line characters as separators. This sample can be used as the starting point for a parser implementation.

i Note: This script sample uses the define function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the require function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType bankStatementParserPlugin
4   */
5
6  define(['N/file', 'N/log'], function(file, log) {
7    return {
8      parseBankStatement: function(context) {
9        var reader = context.input.file.getReader();
10
11        var textUntilFirstComma = reader.readUntil(',');
12        var next10Characters = reader.readChars(10);
13        var textUntilNextNewLine = reader.readUntil('\n');
14        var next100Characters = reader.readChars(100);
15
16        log.debug({
17          title: 'STATEMENT TEXT',
18          details: textUntilFirstComma
19        });
20
21        log.debug({
22          title: 'STATEMENT TEXT',
23          details: next10Characters
24        });
25
26        log.debug({
27          title: 'STATEMENT TEXT',
28          details: textUntilNextNewLine
29        });
30
31        log.debug({
32          title: 'STATEMENT TEXT',
33          details: next100Characters
34        });
35      }
36    }
37  });

```

```
37 | });
```

Read and Log Segments of a File Using a Set of Characters as Separators

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample reads and logs segments from a file using a set of characters as separators.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```
1  /**
2   * @NApiVersion 2.x
3   * @NScriptType bankStatementParserPlugin
4   */
5
6  define(['N/file', 'N/log'], function(file, log) {
7      return {
8          parseBankStatement: function(context) {
9              var statementFile = context.input.file;
10
11              var statementSegmentIterator = statementFile.getSegments({separator: '\\\\|/'}).iterator();
12              statementSegmentIterator.each(function (segment) {
13                  log.debug({
14                      title: 'STATEMENT TEXT',
15                      details: segment.value
16                  });
17                  return true;
18              });
19          }
20      }
21  });
```

N/format Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/format Module](#):

- [Format a Number as a String](#)
- [Format Time of Day as a String](#)
- [Parse a String to a Date Object](#)
- [Parse a String to a Number](#)

Format a Number as a String

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats a raw number value (formatted according to the user's preference) as a string using `format.format(options)`.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/format'],
6    function(format){
7      function formatToString() {
8        // Assume number format is 1.000.000,00 and negative format is (100)
9        var rawNum2 = -44444.44
10       return format.format({value:rawNum2, type: format.Type.FLOAT})
11     }
12     var formattedNum2 = formatToString(); // "44.444,44" -- a string
13   });

```

Format Time of Day as a String

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats the time of day as a string using `format.format(options)`.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/format'],
6    function(format){
7      function formatTimeOfDay() {
8        // Assume the time format is hh:mm (24 hours)
9        var now = new Date(); // Say it's 7:01PM right now.
10       return format.format({value: now, type: format.Type.TIMEOFDAY})
11     }
12     var formattedTime = formatTimeOfDay(); // "19:01" -- a string
13   });

```

Parse a String to a Date Object

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample parses a string (formatted according to the user's preferences) to a raw `Date` object, and then parses it back to the formatted string. This sample uses `format.parse(options)` and `format.format(options)`.

This sample assumes the Date Format set in the preferences is MM/DD/YYYY. You may need to change the value for the date used in this script to match the preferences set in your account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/ui/serverWidget', 'N/format'], function(serverWidget, format) {
6      function parseAndFormatDateString() {
7          // Assuming Date format is MM/DD/YYYY
8          var initialFormattedDateString = "07/28/2015";
9          var parsedDateStringAsRawDateObject = format.parse({
10             value: initialFormattedDateString,
11             type: format.Type.DATE
12         });
13         var formattedDateString = format.format({
14             value: parsedDateStringAsRawDateObject,
15             type: format.Type.DATE
16         });
17         return [parsedDateStringAsRawDateObject, formattedDateString];
18     }
19     function onRequest(context) {
20         var data = parseAndFormatDateString();
21
22         var form = serverWidget.createForm({
23             title: "Date"
24         });
25
26         var fldDate = form.addField({
27             type: serverWidget.FieldType.DATE,
28             id: "date",
29             label: "Date"
30         });
31         fldDate.defaultValue = data[0];
32
33         var fldString = form.addField({
34             type: serverWidget.FieldType.TEXT,
35             id: "dateastext",
36             label: "Date as text"
37         });
38         fldString.defaultValue = data[1];
39
40         context.response.writePage(form);
41     }
42     return {
43         onRequest: onRequest
44     };
45 });

```

Parse a String to a Number

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample parses a string (formatted according to the user's preference) to a raw number value, using `format.parse(options)`.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**

```

```

2  * @ApiVersion 2.x
3  */
4
5  require(['N/format'],
6    function(format){
7      function parseToValue() {
8        // Assume number format is 1.000.000,00 and negative format is -100
9        var formattedNum = "-20.000,25"
10       return format.parse({value:formattedNum, type: format.Type.FLOAT})
11     }
12     var rawNum = parseToValue(); // -20000.25 -- a number
13   });

```

N/format/i18n Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/format/i18n Module](#):

- [Format 12345 as a German String](#)
- [Format a Number as a String Using N/format/i18n](#)
- [Format Currency Based on the Locale Parameter](#)
- [Format Numbers and Currencies Based on the English-India Locale Parameter](#)
- [Format Numbers as Currency Strings](#)
- [Format Numbers Based on the Locale Parameter](#)
- [Parse a Czech Republic Phone Number](#)
- [Parse a U.S. Phone Number](#)

Format 12345 as a German String

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample spells out the number 12345 as a string in German, “zwölftausenddreihundertfünf-undvierzig”.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @ApiVersion 2.x
3   */
4
5   require(['N/format/i18n'],
6     function(format) {
7       var spellOut = format.spellOut({
8         number: 12345,
9         locale: "DE"
10      });
11    });

```



```

12 |     log.debug(spellOut);
13 | });

```

Format a Number as a String Using N/format/i18n

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats a number as a string.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/format/i18n'],
6    function(format) {
7      log.debug("Test of default number formatter:");
8      var numberFormatter = format.getNumberFormatter();
9
10     var gs = numberFormatter.groupSeparator;
11     log.debug("Group separator: " + gs);
12
13     var ds = numberFormatter.decimalSeparator;
14     log.debug("Decimal separator: " + ds);
15
16     var precision = numberFormatter.precision;
17     log.debug("Precision: " + precision);
18
19     var nnf = numberFormatter.negativeNumberFormat;
20     log.debug("Negative Number Format: " + nnf);
21
22     log.debug(numberFormatter.format({number: 12.53}));
23     log.debug(numberFormatter.format({number: 12845.22}));
24     log.debug(numberFormatter.format({number: -5421}));
25     log.debug(numberFormatter.format({number: 0.00}));
26     log.debug(numberFormatter.format({number: 0.3456789}));
27 });

```

Format Currency Based on the Locale Parameter

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats currency based on the locale parameter.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1 | require(['N/format/i18n'], function(format) {

```

```

2   var curFormatter = format.getCurrencyFormatter({
3       locale: "ar_SA"
4   });
5
6   var curCur = curFormatter.currency;
7   log.debug("Currency: " + curCur);
8
9   var sym = curFormatter.symbol;
10  log.debug("Currency symbol: " + sym);
11
12  var numberFormat = curFormatter.numberFormatter;
13
14  var gs = numberFormat.groupSeparator;
15  log.debug("Group separator: " + gs);
16
17  var ds = numberFormat.decimalSeparator;
18  log.debug("Decimal separator: " + ds);
19
20  var prec = numberFormat.precision;
21  log.debug("Precision: " + prec);
22
23  var nnf = numberFormat.negativeNumberFormat;
24  log.debug("Negative Number Format: " + nnf);
25
26  log.debug(curFormatter.format({number: 123456.55}));
27  log.debug(curFormatter.format({number: -123456.55}));
28  });

```

Format Numbers and Currencies Based on the English-India Locale Parameter

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats numbers and currencies based on the English-India (en_IN) locale parameter.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  require(['N/format/i18n'],
2      function(format) {
3      var curFormatter = format.getCurrencyFormatter({locale: "en_IN"});
4
5      var curCur = curFormatter.currency;
6      log.debug("Currency: " + curCur);
7
8      var numberFormat = curFormatter.numberFormatter;
9
10     var sym = curFormatter.symbol;
11     log.debug("Currency symbol: " + sym);
12
13     var gs = numberFormat.groupSeparator;
14     log.debug("Group separator: " + gs);
15
16     var ds = numberFormat.decimalSeparator;
17     log.debug("Decimal separator: " + ds);
18
19     var prec = numberFormat.precision;
20     log.debug("Precision: " + prec);
21

```

```

22     var nnf = numberFormat.negativeNumberFormat;
23     log.debug("Negative Number Format: " + nnf);
24
25     log.debug(curFormatter.format({number: 12345678.55}));
26     log.debug(curFormatter.format({number: -345678.55}));
27 });

```

Format Numbers as Currency Strings

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats numbers as currency strings.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/format/i18n'],
6    function(format) {
7      log.debug("Test of currency formatter - EUR:");
8      var curFormatter = format.getCurrencyFormatter({currency: "EUR"});
9
10     var curCur = curFormatter.currency;
11     log.debug("Currency: " + curCur);
12
13     var numberFormat = curFormatter.numberFormatter;
14
15     var cur3 = curFormatter.symbol;
16     log.debug("Currency symbol: " + cur3);
17
18     var c4 = numberFormat.groupSeparator;
19     log.debug("Group separator: " + c4);
20
21     var c5 = numberFormat.decimalSeparator;
22     log.debug("Decimal separator: " + c5);
23
24     var c6 = numberFormat.precision;
25     log.debug("Precision: " + c6);
26
27     var c7 = numberFormat.negativeNumberFormat;
28     log.debug("Negative Number Format: " + c7);
29
30     log.debug(curFormatter.format({number: 12.53}));
31     log.debug(curFormatter.format({number: -5421}));
32     log.debug(curFormatter.format({number: 0.00}));
33     log.debug(curFormatter.format({number: 0.3456789}));
34 });

```

Format Numbers Based on the Locale Parameter

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample formats numbers based on the locale parameter.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```
1 require(['N/format/i18n'], function(format) {
2     var numberFormatter = format.getNumberFormatter({locale: "fr_FR"});
3     var gs = numberFormatter.groupSeparator;
4     log.debug("Group separator: " + gs);
5
6     var ds = numberFormatter.decimalSeparator;
7     log.debug("Decimal separator: " + ds);
8
9     var prec = numberFormatter.precision;
10    log.debug("Precision: " + prec);
11
12    var nnf = numberFormatter.negativeNumberFormat;
13    log.debug("Negative Number Format: " + nnf);
14
15    log.debug(numberFormatter.format({number: 123456.55}));
16    log.debug(numberFormatter.format({number: -123456.55}));
17 });
```

Parse a Czech Republic Phone Number

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample parses a Czech Republic phone number and logs the resulting country code, extension, national number, number of leading zeros, carrier code, and raw input.

Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```
1 require(['N/format/i18n'], function(format) {
2     var origNumberStr = "602547854ext.154";
3     log.debug("Original number is " + origNumberStr);
4
5     var pnParser = format.getPhoneNumberParser({
6         defaultCountry: format.Country.CZECH_REPUBLIC
7     });
8     var phoneNumber = pnParser.parse({
9         number: origNumberStr
10    });
11
12    log.debug("Country code: " + phoneNumber.countryCode);
13    log.debug("Extension: " + phoneNumber.extension);
14    log.debug("National number: " + phoneNumber.nationalNumber);
15    log.debug("Number of leading zeros: " + phoneNumber.numberOfLeadingZeros);
16    log.debug("Carrier code: " + phoneNumber.carrierCode);
17    log.debug("Raw input: " + phoneNumber.rawInput);
18
19    log.debug("\n-----\nFormatting back:");
20
21    var pnFormatter = format.getPhoneNumberFormatter({});
22    var strNumber = pnFormatter.format({
23        number: phoneNumber
24    });
25    log.debug(strNumber);
26 });
```

Parse a U.S. Phone Number

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample parses a U.S. phone number and logs the country code, extensions, national number, number of leading zeros, carrier code, and raw input.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  require(['N/format/i18n'], function(format) {
2      var origNumberStr = "7524105210ext.154";
3      log.debug("Original number is " + origNumberStr);
4
5      var pnParser = format.getPhoneNumberParser({
6          defaultCountry: format.Country.UNITED_STATES
7      });
8      var phoneNumber = pnParser.parse({
9          number: origNumberStr
10     });
11
12     log.debug(phoneNumber.countryCode);
13     log.debug(phoneNumber.extension);
14     log.debug(phoneNumber.nationalNumber);
15     log.debug(phoneNumber.numberofLeadingZeros);
16     log.debug(phoneNumber.carrierCode);
17     log.debug(phoneNumber.rawInput);
18
19     var pnFormatter = format.getPhoneNumberFormatter({
20         formatType: format.PhoneNumberFormatType.NATIONAL
21     });
22     var strNumber = pnFormatter.format({
23         number: phoneNumber
24     });
25     log.debug(strNumber);
26 });

```

N/http Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/http Module](#):

- [Redirect a New Sales Order and Set the Entity Field](#)
- [Request a URL Using `http.get`](#)

Redirect a New Sales Order and Set the Entity Field

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a Suitelet to redirect to a new sales order record and set the entity field (which represents the customer).

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: The value used in this sample for the entity field is a placeholder. Before using this sample, replace the entity field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5
6   // This script redirects a new sales order record and sets the entity.
7   define(['N/http'], (record, http)=> {
8       function onRequest(context) {
9           context.response.sendRedirect({
10               type: http.RedirectType.RECORD,
11               identifier: record.Type.SALES_ORDER,
12               parameters: ({
13                   entity: 6
14               })
15           });
16       }
17       return {
18           onRequest: onRequest
19       };
20   });

```

Request a URL Using `http.get`

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use an HTTP GET request for a URL.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5   // This script uses an HTTP GET request for a URL.
6   require(['N/http'], (http)=> {
7       function sendGetRequest() {
8           let response = http.get({
9               url: 'http://www.google.com'
10            });
11       }
12       sendGetRequest();
13   });

```

N/https Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/https Module](#):

- [Concatenate API Secrets with Strings](#)
- [Create a Form with a Field that Generates a GUID](#)
- [Create a JWT Token Using a Secure String](#)
- [Create an Authentication Header Using a Secure String](#)
- [Generate a Secure Token and a Secret Key](#)

Concatenate API Secrets with Strings

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to concatenate a string value to use as an API secret. API Secrets are string values that cannot be concatenated directly. In some cases, a code-generated string (for example, date stamp, account ID, sequence numbers) needs to be merged with the API secret. To merge the values, the N/https module must be imported on the script file and use the `createSecureString()` API to initialize both secret API values and ordinary string.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

⚠ Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5   // This script uses appendSecureString to concatenate strings to use as an API secre.
6   require(['N/https', 'N/runtime'], (https, runtime) => {
7     function concatToCreateSecureString() {
8       let baseUrl = https.createSecureString({
9         input: 'www.someurl.com/add?apikey='
10      });
11      let apiKey = https.createSecureString({
12        input: '{CUSTSECRET_SOME_INTEGRATION}'
13      });
14      let url = baseUrl.appendSecureString({
15        secureString: apiKey
16      });
17    }
18    concatToCreateSecureString();
19  });

```

Create a JWT Token Using a Secure String

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a JWT token using `https.SecureString`. For more information about `SecureString`, see the help topic [https.SecureString](#).

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType Suitelet
4   */
5
6   // This script creates a JWT token using https.SecureString.
7   define(['/N/https', 'N/encode'], (https, encode) => {
8       function onRequest(context) {
9           let nameToken = "custsecret_myName";
10          let passwordToken = "custsecret_mypPassword";
11          let headerObj = {
12              "alg": "HS256",
13              "typ": "JWT"
14          }
15          let payloadObj = {
16              "sub": "1234567890",
17              "name": "John Doe",
18              "iat": 1516239002
19          }
20
21          let headerJSON = JSON.stringify(headerObj);
22          let payloadJSON = JSON.stringify(payloadObj);
23          let headerBASE64 = encode.convert({
24              string: headerJSON,
25              inputEncoding: encode.Encoding.UTF_8,
26              outputEncoding: encode.Encoding.BASE_64_URL_SAFE
27          });
28
29          let payloadBASE64 = encode.convert({
30              string: payloadJSON,
31              inputEncoding: encode.Encoding.UTF_8,
32              outputEncoding: encode.Encoding.BASE_64_URL_SAFE
33          });
34
35          let headerBASE64 = headerBASE64.replace(/=/g, ""); // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64 without
padding
36          let payloadBASE64 = payloadBASE64.replace(/=/g, ""); // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64
without padding
37
38          let secStringJwtSignature = https.createSecureString({
39              input: headerBASE64 + "." + payloadBASE64
40          })
41          .hmac({
42              algorithm: https.HashAlg.SHA256,
43              key: https.createSecretKey({
44                  secret: passwordToken,
45                  encoding: encode.Encoding.UTF_8
46              }),
47              resultEncoding: encode.Encoding.BASE_64_URL_SAFE
48          })
49          .replaceString({ // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64 without padding
50              pattern: "=",
51              replacement: ""
52          })
53
54          let secStringJwtAuthHeader = https.createSecureString({
55              input: "Bearer " + headerBASE64 + "." + payloadBASE64 + "."
56          })
57          .appendSecureString({
58              secureString: secStringJwtSignature,
59              keepEncoding: true

```



```

60     })
61
62     // Reflect the response using a echo-request suitelet
63     let resp = https.get({
64         url: "myURL",
65         headers: {
66             "Authorization": secStringJwtAuthHeader
67         }
68     });
69
70     {
71         log.error("resp-code", resp.code);
72         log.debug("resp-body", resp.body);
73
74         let respAuth = JSON.parse(resp.body)["headers"]["Authorization"];
75
76         log.debug("resp-head-auth", respAuth);
77         log.debug("resp-head-auth-expected",
78             "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.uel3RLILSJ9Q9W2Gomh8vAJQAgdbnd6TS4b7plyF0tA" ); // see https://jwt.io/#debugger-io
79     }
80 }
81 return {
82     onRequest: onRequest
83 };
84 });

```

Create a Form with a Field that Generates a GUID

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a Suitelet to create a form field that generates a GUID. For more information about credential fields, see the help topic [Form.addCredentialField\(options\)](#).

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Note: The default maximum length for a credential field is 32 characters. If needed, use the `Field.maxLength` property to change this value.

The values for `restrictToDomains`, `restrictToScriptIds`, and `baseUrl` in this sample are placeholders. You must replace them with valid values from your NetSuite account.

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @ApiVersion 2.1
3   * @NScriptType Suitelet
4   */
5
6   // This script creates a form with a credential field.
7   define(['N/ui/serverWidget', 'N/https', 'N/url'], (ui, https, url) => {
8       function onRequest(context) {
9           if (context.request.method === 'GET') {
10               const form = ui.createForm({
11                   title: 'Password Form'
12               });
13
14               const credField = form.addCredentialField({
15                   id: 'password',
16                   label: 'Password',

```

```

17         restrictToDomains: ['<accountID>.app.netsuite.com'],
18         restrictToCurrentUser: false,
19         restrictToScriptIds: 'customscript_my_script'
20     });
21
22     credField.maxLength = 32;
23
24     form.addSubmitButton();
25
26     context.response.writePage({
27         pageObject: form
28     });
29 }
30 else {
31     // Request to an existing Suitelet with credentials
32     let passwordGuid = context.request.parameters.password;
33
34     // Replace SCRIPTID and DEPLOYMENTID with the internal ID of the suitelet script and deployment in your account
35     let baseUrl = url.resolveScript({
36         scriptId: SCRIPTID,
37         deploymentId: DEPLOYMENTID,
38         returnExternalURL: true
39     });
40
41     let authUrl = baseUrl + '&pwd={' + passwordGuid + '}';
42
43     let secureStringUrl = https.createSecureString({
44         input: authUrl
45     });
46
47     let headers = ({
48         'pwd': passwordGuid
49     });
50
51     let response = https.post({
52         credentials: [passwordGuid],
53         url: secureStringUrl,
54         body: {authorization: ' ' + passwordGuid + ' ', data: 'anything can be here'},
55         headers: headers
56     });
57 }
58 }
59 return {
60     onRequest: onRequest
61 };
62 });

```

Create an Authentication Header Using a Secure String

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a Suitelet to create an authentication header and send the request to a service using an `https.SecureString`; the service requires an authentication header. For more information about `SecureString`, see the help topic [https.SecureString](#).

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

2  * @NApiVersion 2.1
3  * @NScriptType Suitelet
4  */
5
6  // This script creates an authentication header using an https.SecureString.
7  define(['N/https', 'N/encode'], (https, encode) => {
8      function onRequest(context) {
9
10         // Secrets with these two Script IDs must be existing and allowed for this script
11         const nameToken = "custsecret_myName";
12         const passwordToken = "custsecret_mypassword";
13
14         // Create BASE-64 encoded name:password pair
15         const secStringKeyInBase64 = https.createSecureString({
16             input: "{" + nameToken + "}:" + passwordToken + "}"
17         });
18
19         secStringKeyInBase64.convertingEncoding({
20             toEncoding: encode.Encoding.BASE_64,
21             fromEncoding: encode.Encoding.UTF_8
22         });
23
24         // Construct the Authorization header
25         const secStringBasicAuthHeader = https.createSecureString({
26             input: "Basic "
27         });
28
29         secStringBasicAuthHeader.appendSecureString({
30             secureString: secStringKeyInBase64,
31             keepEncoding: true
32         });
33
34         // Send the request to third party with the Authorization header
35         const resp = https.get({
36             url: "myUrl",
37             headers: {
38                 "Authorization": secStringBasicAuthHeader
39             }
40         });
41
42         // Log the response code
43         log.debug("resp-code", resp.code);
44     };
45     return {
46         onRequest: onRequest
47     };
48 });

```

Generate a Secure Token and a Secret Key

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a GUID to generate a secure token and a secret key. To run this sample in the debugger, you must replace the GUID with one specific to your account.

i Note: This sample script uses the require function so that you can copy it into the SuiteScript Debugger and test it. You must use the define function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2  * @NApiVersion 2.1

```

```

3  */
4
5  // This script uses a GUID to generate a secure token and a secret key.
6  require(['N/https', 'N/runtime'], (https, runtime) => {
7      function createSecureString() {
8          const passwordGuid = '{284CFB2D225B1D76FB94D150207E49DF}';
9          let secureToken = https.createSecureString({
10             input: passwordGuid
11         });
12         let secretKey = https.createSecretKey({
13             input: passwordGuid
14         });
15         secureToken = secureToken.hmac({
16             algorithm: https.HashAlg.SHA256,
17             key: secretKey
18         });
19     }
20     createSecureString();
21 });

```

N/https/clientCertificate Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/https/clientCertificate Module](#):

- [Send a Secure Post Request to a Remote URL](#)

Send a Secure Post Request to a Remote URL

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample sends a secure post request to a remote URL.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5  require(['N/https/clientCertificate'], (cert) => {
6      // Set the URL
7      const url = "https://nfe.fazenda.sp.gov.br/ws/cadconsultacadastro4.asmx";
8
9      let data = "<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope xmlns:soapenv='http://www.w3.org/2003/05/soap-envelope'><soapenv:Body><ns1:nfeDadosMsg xmlns:ns1='http://www.portalfiscal.inf.br/nfe/wsd1/CadConsultaCadastro4'><ConsCad xmlns='http://www.portalfiscal.inf.br/nfe' versao='2.00'><infCons><xServ>CONS-CAD</xServ><UF>SP</UF><CNPJ>47508411000156</CNPJ></infCons></ConsCad></ns1:nfeDadosMsg></soapenv:Body></soapenv:Envelope>";
10
11     const key = "custcertificate1";
12
13     let headers = {
14         "Content-Type": "application/soap+xml"
15     };
16

```

```

17   let response = cert.post({
18     url: url,
19     certId: key,
20     body: data,
21     headers: headers
22   });
23   log.debug(response.body);
24 })

```

N/keyControl Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/keyControl Module](#):

- [Add a Secret Key Field to a Form](#)
- [Create a Secret Key](#)

Add a Secret Key Field to a Form

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to add a secret key field.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/ui/serverWidget', 'N/file', 'N/keyControl', 'N/runtime'], function(ui, file, keyControl, runtime) {
6     function onRequest(context) {
7         var request = context.request;
8         var response = context.response;
9
10        if (request.method === 'GET') {
11            var form = ui.createForm({
12                title: 'Enter Password'
13            });
14
15            var credField = form.addSecretKeyField({
16                id: 'custfield_password',
17                label: 'Password',
18                restrictToScriptIds: [runtime.getCurrentScript().id],
19                restrictToCurrentUser: true //Depends on use case
20            });
21            credField.maxLength = 64;
22
23            form.addSubmitButton();
24            response.writePage(form);
25        } else {
26            // Read the request parameter matching the field ID we specified in the form
27            var passwordToken = request.parameters.custfield_password;
28
29            var pem = file.load({
30                id: 422
31            });
32        }
33    }
34 }

```

```

33     var key = keyControl.createKey();
34     key.file = pem;
35     key.name = 'Test';
36     key.password = passwordToken;
37     key.save();
38   }
39 }
40 return {
41   onRequest: onRequest
42 };
43 });

```

Create a Secret Key

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a key.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/keyControl', 'N/file'], function(keyControl, file){
6     var key = keyControl.createKey();
7     key.file = file.load(422);
8     //id of file containing private key (id_ecdsa or id_rsa)
9     key.name = "SFTP key";
10    key.save();
11  })

```

N/log Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/log Module](#):

- [Create Debug Log Messages](#)

Create Debug Log Messages

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create each type of log messages.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**

```

```

2  * @NApiVersion 2.x
3  * @NScriptType UserEventScript
4  */
5
6  // This script creates each type of log message.
7  define(['N/log'],function(log) {
8      function beforeLoad(context) {
9          var myValue = 'value';
10
11          var myObject = {
12              name: 'Jane',
13              id: '123'
14          };
15
16          // An audit log message
17          log.audit({
18              title: 'Audit Entry',
19              details: myObject
20          });
21
22          // A debug log message
23          log.debug({
24              title: 'Debug Entry',
25              details: 'Value of myValue is: ' + myValue
26          });
27
28          // An emergency log message
29          log.emergency({
30              title: 'Emergency Entry',
31              details: 'Value of myValue is: ' + myValue
32          });
33
34          // An error log message
35          log.error({
36              title: 'Error Entry',
37              details: 'Value of myValue is: ' + myValue
38          });
39      }
40      return {
41          beforeLoad: beforeLoad
42      };
43  });

```

N/piremoval Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/piremoval Module](#):

- [Remove Phone Numbers and Comments from Customer Records](#)

Remove Phone Numbers and Comments from Customer Records

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to remove the phone numbers and comments in specific customer records from the record fields (field values), system notes, and workflow history. The removed values are replaced with **removed_value**.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/piremoval'], function(piremoval) {
6      function removePersonalInformation() {
7          var piRemovalTask = piremoval.createTask({
8              recordType: 'customer',
9              recordIds: [11, 19],
10             fieldIds: ['comments', 'phone'],
11             workflowIds: [1],
12             historyOnly: false,
13             historyReplacement: 'removed_value'
14         });
15
16         piRemovalTask.save();
17         var taskId = piRemovalTask.id;
18
19         var piRemovalTaskInProgress = piremoval.loadTask(taskId);
20         piRemovalTaskInProgress.run();
21
22         var status = piremoval.getTaskStatus(taskId);
23     };
24
25     removePersonalInformation();
26 });

```

N/plugin Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/plugin Module](#):

Find Plug-in Implementations

Find Plug-in Implementations

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows an implementation of a custom plug-in interface. To test this sample, you need a custom plug-in type with a script ID of `customscript_magic_plugin` and an interface with a single method, `int doTheMagic(int, int)`.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType pluginTypeimpl
4   */
5  define(function() {

```



```

6      return {
7          doTheMagic: function(operand1, operand2) {
8              return operand1 + operand2;
9          }
10     };
11 });

```

The following Suitelet iterates through all implementations of the custom plug-in type `customscript_magic_plugin`. For the plug-in to be recognized, the Suitelet script record must specify the plug-in type on the Custom Plug-in Types subtab.

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/plugin'], function(plugin) {
6      function onRequest(context) {
7          var impls = plugin.findImplementations({
8              type: 'customscript_magic_plugin'
9          });
10
11         for (var i = 0; i < impls.length; i++) {
12             var pl = plugin.loadImplementation({
13                 type: 'customscript_magic_plugin',
14                 implementation: impls[i]
15             });
16             log.debug('impl ' + impls[i] + ' result = ' + pl.doTheMagic(10, 20));
17         }
18
19         var pl = plugin.loadImplementation({
20             type: 'customscript_magic_plugin'
21         });
22         log.debug('default impl result = ' + pl.doTheMagic(10, 20));
23     }
24
25     return {
26         onRequest: onRequest
27     };
28 });

```

N/portlet Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/portlet Module](#):

- [Create a Form Portlet with a Button That Allows User Adjustments](#)

Create a Form Portlet with a Button That Allows User Adjustments

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a form portlet that allows users to adjust its height and width. It creates two text fields representing the height and width of the portlet, measured in pixels. It also creates a button that runs the `portlet.resize()` method to adjust the height and width of the portlet based on the values of the text fields.

This sample also shows how to create a button that uses the `portlet.refresh()` method. When the button is clicked, the portlet is updated to show the current date.

For more information about how a portlet is displayed on the NetSuite dashboard, see the help topic [SuiteScript 2.x Portlet Script Type](#).

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Portlet
4   * @NScriptPortletType form
5   */
6
7  define([], function() {
8      function render(context) {
9          var portletObj = context.portlet;
10         portletObj.title = 'Test Form Portlet';
11         setComponentsForResize();
12         setComponentsForRefresh();
13
14         function setComponentsForResize() {
15             var DEFAULT_HEIGHT = '50';
16             var DEFAULT_WIDTH = '50';
17             var inlineHTMLField = portletObj.addField({
18                 id: 'divfield',
19                 type: 'inlinehtml',
20                 label: 'Test inline HTML'
21             });
22             inlineHTMLField.defaultValue = '<div id=\''divfield_elem\'' style=\'border: 1px dotted red; height: '
23             + DEFAULT_HEIGHT + 'px; width: ' + DEFAULT_WIDTH + 'px;\></div>';
24             inlineHTMLField.updateLayoutType({
25                 layoutType: 'normal'
26             });
27             inlineHTMLField.updateBreakType({
28                 breakType: 'startcol'
29             });
30             var resizeHeight = portletObj.addField({
31                 id: 'resize_height',
32                 type: 'text',
33                 label: 'Resize Height'
34             });
35             resizeHeight.defaultValue = DEFAULT_HEIGHT;
36             var resizeWidth = portletObj.addField({
37                 id: 'resize_width',
38                 type: 'text',
39                 label: 'Resize Width'
40             });
41             resizeWidth.defaultValue = DEFAULT_WIDTH;
42             var resizeLink = portletObj.addField({
43                 id: 'resize_link',
44                 type: 'inlinehtml',
45                 label: 'Resize link'
46             });
47             resizeLink.defaultValue = resizeLink.defaultValue = '<a id=\''resize_link\'' onclick=\''require([\'SuiteScripts/portletApiTestHelper\'], function(portletApiTestHelper) {portletApiTestHelper.resizePortlet(); }) \' href=\''#\''>Resize</a><br>';
48         }
49
50         function setComponentsForRefresh() {
51             var textField = portletObj.addField({
52                 id: 'refresh_output',
53                 type: 'text',
54                 label: 'Date.now().toString()'
55             });
56             textField.defaultValue = Date.now().toString();
57             var refreshLink = portletObj.addField({
58                 id: 'refresh_link',
59                 type: 'inlinehtml',
60                 label: 'Refresh link'
61             });
62         }
63     }
64 });

```

```

61         refreshLink.defaultValue = '<a id=\''refresh_link\'' onclick=\''require(["SuiteScripts/portletApiTestHelper"],
function(portletApiTestHelper) {portletApiTestHelper.refreshPortlet(); }) \' href=\'#\''>Refresh</a>';
62     }
63 }
64
65 return {
66     render: render
67 };
68 })
69
70 // portletApiTestHelper.js
71 define(['N/portlet'], function(portlet) {
72     function refreshPortlet() {
73         portlet.refresh();
74     }
75
76     function resizePortlet() {
77         var div = document.getElementById('divfield_elem');
78         var newHeight = parseInt(document.getElementById('resize_height').value);
79         var newWidth = parseInt(document.getElementById('resize_width').value);
80         div.style.height = newHeight + 'px';
81         div.style.width = newWidth + 'px';
82         portlet.resize();
83     }
84
85     return {
86         refreshPortlet: refreshPortlet,
87         resizePortlet: resizePortlet
88     };
89 });

```

N/query Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/query Module](#):

- [Create a Query for a Custom Field](#)
- [Create a Query for Customer Records and Run It as a Non-Paged Query](#)
- [Create a Query for Transaction Records and Run It as a Paged Query](#)
- [Convert a Query to a SuiteQL and Run It](#)
- [Create a Query Using a Specific Record Field](#)
- [Run an Arbitrary SuiteQL Query](#)

Create a Query for a Custom Field

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a query for a custom field, `custrecord_my_custom_field`, and obtains the internal ID of the field.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /*
2   * @NApiVersion 2.x
3   */
4  require(['N/query'], function(query) {
5      var customFieldIdQuery = query.create({

```

```

6         type: query.Type.CUSTOM_FIELD
7     });
8     customFieldIdQuery.columns = [
9         customFieldIdQuery.createColumn({
10             fieldId: 'internalid'
11         })
12     ];
13     customFieldIdQuery.condition = customFieldIdQuery.createCondition({
14         fieldId: 'scriptid',
15         operator: query.Operator.IS,
16         values: 'custrecord_my_custom_field'
17     });
18
19     var results = customFieldIdQuery.run().asMappedResults();
20     var customFieldInternalId = results[0].internalid;
21     log.debug(customFieldInternalId);
22 });

```

Create a Query for Customer Records and Run It as a Non-Paged Query

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a query for customer records, joins the query with two other query types, and runs the query.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  require(['N/query'], query => {
5      // Create a query definition for customer records
6      let myCustomerQuery = query.create({
7          type: query.Type.CUSTOMER
8      });
9
10     // Join the original query definition based on the salesrep field. In a customer
11     // record, the salesrep field contains a reference to an employee record. When you
12     // join based on this field, you are joining the query definition with the employee
13     // query type, and you can access the fields of the joined employee record in
14     // your query.
15     let mySalesRepJoin = myCustomerQuery.autoJoin({
16         fieldId: 'salesrep'
17     });
18
19     // Join the joined query definition based on the location field. In an employee
20     // record, the location field contains a reference to a location record.
21     let myLocationJoin = mySalesRepJoin.autoJoin({
22         fieldId: 'location'
23     });
24
25     // Create conditions for the query
26     let firstCondition = myCustomerQuery.createCondition({
27         fieldId: 'id',
28         operator: query.Operator.EQUAL,
29         values: 107
30     });
31     let secondCondition = myCustomerQuery.createCondition({

```

```

32     fieldId: 'id',
33     operator: query.Operator.EQUAL,
34     values: 2647
35   });
36   let thirdCondition = mySalesRepJoin.createCondition({
37     fieldId: 'email',
38     operator: query.Operator.START_WITH_NOT,
39     values: 'example'
40   });
41
42   // Combine conditions using and() and or() operator methods. In this example,
43   // the combined condition states that the id field of the customer record must
44   // have a value of either 107 or 2647, and the email field of the employee
45   // record (the record that is referenced in the salesrep field of the customer
46   // record) must not start with 'example'.
47   myCustomerQuery.condition = myCustomerQuery.and(
48     thirdCondition, myCustomerQuery.or(firstCondition, secondCondition)
49   );
50
51   // Create query columns
52   myCustomerQuery.columns = [
53     myCustomerQuery.createColumn({
54       fieldId: 'entityid'
55     }),
56     myCustomerQuery.createColumn({
57       fieldId: 'id'
58     }),
59     mySalesRepJoin.createColumn({
60       fieldId: 'entityid'
61     }),
62     mySalesRepJoin.createColumn({
63       fieldId: 'email'
64     }),
65     mySalesRepJoin.createColumn({
66       fieldId: 'hiredate'
67     }),
68     myLocationJoin.createColumn({
69       fieldId: 'name'
70     })
71   ];
72
73   // Sort the query results based on query columns
74   myCustomerQuery.sort = [
75     myCustomerQuery.createSort({
76       column: myCustomerQuery.columns[3]
77     }),
78     myCustomerQuery.createSort({
79       column: myCustomerQuery.columns[0],
80       ascending: false
81     })
82   ];
83
84   // Run the query
85   let resultSet = myCustomerQuery.run();
86
87   // Retrieve and log the results
88   let results = resultSet.results;
89   for (let i = results.length - 1; i >= 0; i--)
90     log.debug(results[i].values);
91   log.debug(resultSet.types);
92   });

```

Create a Query for Transaction Records and Run It as a Paged Query

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a query for transaction records, joins the query with another query type, and runs the query as a paged query.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/query'], query => {
5       // Create a query definition for transaction records
6       let myTransactionQuery = query.create({
7           type: query.Type.TRANSACTION
8       });
9
10      // Join the original query definition based on the employee field. In a transaction
11      // record, the employee field contains a reference to an employee record. When you
12      // join based on this field, you are joining the query definition with the employee
13      // query type, and you can access the fields of the joined employee record in
14      // your query.
15      let myEmployeeJoin = myTransactionQuery.autoJoin({
16          fieldId: 'employee'
17      });
18
19      // Create a condition for the transaction query
20      let transactionCondition = myTransactionQuery.createCondition({
21          fieldId: 'isreversal',
22          operator: query.Operator.IS,
23          values: true
24      });
25      myTransactionQuery.condition = transactionCondition;
26
27      // Create a query column
28      myTransactionQuery.columns = [
29          myEmployeeJoin.createColumn({
30              fieldId: 'subsidiary'
31          })
32      ];
33
34      // Sort the query results based on a query column
35      myTransactionQuery.sort = [
36          myTransactionQuery.createSort({
37              column: myTransactionQuery.columns[0],
38              ascending: false
39          })
40      ];
41
42      // Run the query as a paged query with 10 results per page
43      let results = myTransactionQuery.runPaged({
44          pageSize: 10
45      });
46
47      log.debug(results.pageRanges.length);
48      log.debug(results.count);
49  }

```

```

50 // Retrieve the query results using an iterator
51 let iterator = results.iterator();
52 iterator.each(function(result) {
53     let page = result.value;
54     log.debug(page.pageRange.size);
55     return true;
56 })
57
58 // Alternatively, retrieve the query results by looping through
59 // each result
60 for (let i = 0; i < results.pageRanges.length; i++) {
61     let page = results.fetch(i);
62     log.debug(page.pageRange.size);
63 }
64 });

```

Convert a Query to a SuiteQL and Run It

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a query for customer records, converts it to its SuiteQL representation, and runs it.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4  require(['N/query'], function(query) {
5      var myCustomerQuery = query.create({
6          type: query.Type.CUSTOMER
7      });
8
9      myCustomerQuery.columns = [
10         myCustomerQuery.createColumn({
11             fieldId: 'entityid'
12         }),
13         myCustomerQuery.createColumn({
14             fieldId: 'email'
15         })
16     ];
17
18     myCustomerQuery.condition = myCustomerQuery.createCondition({
19         fieldId: 'isperson',
20         operator: query.Operator.IS,
21         values: [true]
22     });
23
24     var mySQLCustomerQuery = myCustomerQuery.toSuiteQL();
25
26     var results = mySQLCustomerQuery.run();
27 });

```

Create a Query Using a Specific Record Field

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a query for records using the value of the `operationdisplaytext` field. If you run this sample in your account, be sure to replace the placeholder value `<transactionId>` with a valid value from your account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2  * @NApiVersion 2.x
3  */
4  require(['N/query'], function(query) {
5      var mfgComponent = query.create({
6          type: query.Type.MANUFACTURING_COMPONENT
7      });
8
9      var mfgOperation = mfgComponent.autoJoin({
10         fieldId: 'operationdisplaytext'
11     });
12
13     mfgComponent.columns = [
14         mfgComponent.createColumn({
15             fieldId: 'operationdisplaytext'
16         }),
17         mfgComponent.createColumn({
18             fieldId: 'item'
19         }),
20         mfgOperation.createColumn({
21             fieldId: 'operationsequence'
22         })
23     ];
24
25     mfgComponent.condition = mfgComponent.and(
26         mfgComponent.createCondition({
27             fieldId: 'transaction',
28             operator: query.Operator.ANY_OF,
29             values: <transactionId>
30         }),
31         mfgOperation.createCondition({
32             fieldId: 'operationsequence',
33             operator: query.Operator.EQUAL,
34             values: 20
35         })
36     );
37
38     var results = mfgComponent.run();
39 });

```

Run an Arbitrary SuiteQL Query

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample constructs a SuiteQL query string, runs the query as a paged query, and iterates over the results.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2  * @NApiVersion 2.x
3  */

```



```

4 require(['N/query'], function(query) {
5     var sql =
6         "SELECT " +
7         "    scriptDeployment.primarykey, scriptexecutioncontextmap.executioncontext " +
8         "FROM " +
9         "    scriptDeployment, scriptexecutioncontextmap " +
10        "WHERE " +
11        "    scriptexecutioncontextmap.scriptrecord = scriptDeployment.primarykey " +
12        "AND " +
13        "    scriptexecutioncontextmap.executioncontext IN ('WEBSTORE', 'WEBAPPLICATION');
14
15    var resultIterator = query.runSuiteQLPaged({
16        query: sql,
17        pageSize: 10
18    }).iterator();
19
20    resultIterator.each(function(page) {
21        var pageIterator = page.value.data.iterator();
22        pageIterator.each(function(row) {
23            log.debug('ID: ' + row.value.getValue(0) + ', Context: ' + row.value.getValue(1));
24            return true;
25        });
26        return true;
27    });
28 });

```

N/record Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/record Module](#):

- [Access Sublists and a Subrecord from a Record](#)
- [Access Sublists and a Subrecord from a Record Asynchronously Using Promise Methods](#)
- [Call a Macro on a Sales Order Record](#)
- [Create and Save a Contact Record](#)
- [Create and Save a Contact Record Asynchronously Using Promise Methods](#)
- [Create Multiple Sales Records Using a Scheduled Script](#)

Access Sublists and a Subrecord from a Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to access sublists and a subrecord from a record. This sample requires the Advanced Number Inventory Management feature.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2  * @NApiVersion 2.x

```

```

3  */
4
5  require(['N/record'], function(record) {
6      function createPurchaseOrder() {
7          var rec = record.create({
8              type: 'purchaseorder',
9              isDynamic: true
10         });
11         rec.setValue({
12             fieldId: 'entity',
13             value: 52
14         });
15         rec.setValue({
16             fieldId: 'location',
17             value: 2
18         });
19         rec.selectNewLine({
20             sublistId: 'item'
21         });
22         rec.setCurrentSublistValue({
23             sublistId: 'item',
24             fieldId: 'item',
25             value: 190
26         });
27         rec.setCurrentSublistValue({
28             sublistId: 'item',
29             fieldId: 'quantity',
30             value: 2
31         });
32         subrecordInvDetail = rec.getCurrentSublistSubrecord({
33             sublistId: 'item',
34             fieldId: 'inventorydetail'
35         });
36         subrecordInvDetail.selectNewLine({
37             sublistId: 'inventoryassignment'
38         });
39         subrecordInvDetail.setCurrentSublistValue({
40             sublistId: 'inventoryassignment',
41             fieldId: 'receiptinventorynumber',
42             value: 'myinventoryNumber'
43         });
44         subrecordInvDetail.commitLine({
45             sublistId: 'inventoryassignment'
46         });
47         subrecordInvDetail.selectLine({
48             sublistId: 'inventoryassignment',
49             line: 0
50         });
51         var myInventoryNumber = subrecordInvDetail.getCurrentSublistValue({
52             sublistId: 'inventoryassignment',
53             fieldId: 'receiptinventorynumber'
54         });
55         rec.commitLine({
56             sublistId: 'item'
57         });
58         var recordId = rec.save();
59     }
60     createPurchaseOrder();
61 });

```

Access Sublists and a Subrecord from a Record Asynchronously Using Promise Methods

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to access sublists and a subrecord from a record using promise methods. This sample requires the Advanced Number Inventory Management feature.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/record'], function(record) {
6      function createPurchaseOrder() {
7          var createRecordPromise = record.create.promise({
8              type: 'purchaseorder',
9              isDynamic: true
10         });
11         createRecordPromise.then(function(rec) {
12             rec.setValue({
13                 fieldId: 'entity',
14                 value: 52
15             });
16             rec.setValue({
17                 fieldId: 'location',
18                 value: 2
19             });
20             rec.selectNewLine({
21                 sublistId: 'item'
22             });
23             rec.setCurrentSublistValue({
24                 sublistId: 'item',
25                 fieldId: 'item',
26                 value: 190
27             });
28             rec.setCurrentSublistValue({
29                 sublistId: 'item',
30                 fieldId: 'quantity',
31                 value: 2
32             });
33             subrecordInvDetail = rec.getCurrentSublistSubrecord({
34                 sublistId: 'item',
35                 fieldId: 'inventorydetail'
36             });
37             subrecordInvDetail.selectNewLine({
38                 sublistId: 'inventoryassignment'
39             });
40             subrecordInvDetail.setCurrentSublistValue({
41                 sublistId: 'inventoryassignment',
42                 fieldId: 'receiptinventorynumber',
43                 value: 'myinventoryNumber'
44             });
45             subrecordInvDetail.commitLine({
46                 sublistId: 'inventoryassignment'
47             });
48             subrecordInvDetail.selectLine({
49                 sublistId: 'inventoryassignment',
50                 line: 0
51             });
52             var myInventoryNumber = subrecordInvDetail.getCurrentSublistValue({
53                 sublistId: 'inventoryassignment',
54                 fieldId: 'receiptinventorynumber'
55             });
56             rec.commitLine({
57                 sublistId: 'item'
58             });

```

```

59     var recordId = rec.save();
60     }, function(err) {
61         log.error('Unable to create purchase order!', err.name);
62     });
63 }
64 createPurchaseOrder();
65 });

```

Call a Macro on a Sales Order Record

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows you how to call a `calculateTax` macro on a sales order record. To execute a macro on a record, the record must be created or loaded in dynamic mode. Note that the SuiteTax feature must be enabled to successfully execute the macro used in this sample.

For information about record macros, see the help topic [Overview of Record Action and Macro APIs](#).

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/record'],
6      function(record) {
7
8      var recordObj = record.create({
9          type: record.Type.SALES_ORDER,
10         isDynamic: true
11     });
12
13     var ENTITY_VALUE = 1;
14     var ITEM_VALUE = 1;
15     recordObj.setValue({
16         fieldId: 'entity',
17         value: ENTITY_VALUE
18     });
19     recordObj.selectNewLine({
20         sublistId: 'item'
21     });
22     recordObj.setCurrentSublistValue({
23         sublistId: 'item',
24         fieldId: 'item',
25         value: ITEM_VALUE
26     });
27     recordObj.setCurrentSublistValue({
28         sublistId: 'item',
29         fieldId: 'quantity',
30         value: 1
31     });
32     recordObj.commitLine({
33         sublistId: 'item'
34     });
35
36     var totalBeforeTax = recordObj.getValue({fieldId: 'total'});
37
38     // get macros available on the record
39     var macros = recordObj.getMacros();
40
41     // execute the macro

```

```

42 |     if ('calculateTax' in macros)
43 |     {
44 |         macros.calculateTax(); // For promise version use: macros.calculateTax.promise()
45 |     }
46 |     // Alternative (direct) macro execution
47 |     // var calculateTax = recordObj.getMacro({id: 'calculateTax'});
48 |     // calculateTax(); // For promise version use: calculateTax.promise()
49 |     var totalAfterTax = recordObj.getValue({fieldId: 'total'});
50 |
51 |     var recordId = recordObj.save({
52 |         enableSourcing: false,
53 |         ignoreMandatoryFields: false
54 |     });
55 | });

```

Create and Save a Contact Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use the N/record module to create and save a contact record.

i Note: This sample script uses the require function so that you can copy it into the SuiteScript Debugger and test it. You must use the define function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: Some of the values in these samples are placeholders. Before using these samples, replace all hard-coded values, such as IDs and file paths, with valid values from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1 | /**
2 |  * @NApiVersion 2.1
3 |  */
4 | require(['N/record'], record => {
5 |     // Create an object to hold name data for the contact
6 |     const nameData = {
7 |         firstname: 'John',
8 |         middlename: 'Doe',
9 |         lastname: 'Smith'
10 |    };
11 |
12 |    // Create a contact record
13 |    let objRecord = record.create({
14 |        type: record.Type.CONTACT,
15 |        isDynamic: true
16 |    });
17 |
18 |    // Set the values of the subsidiary, firstname, middlename,
19 |    // and lastname properties
20 |    objRecord.setValue({
21 |        fieldId: 'subsidiary',
22 |        value: '1'
23 |    });
24 |    for (let key in nameData) {
25 |        if (nameData.hasOwnProperty(key)) {
26 |            objRecord.setValue({
27 |                fieldId: key,
28 |                value: nameData[key]

```

```

29     });
30   }
31 }
32
33 // Save the record
34 let recordId = objRecord.save({
35   enableSourcing: false,
36   ignoreMandatoryFields: false
37 });
38 });

```

Create and Save a Contact Record Asynchronously Using Promise Methods

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create and save a contact record using promise methods.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

i Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType ClientScript
4   */
5   require(['N/record'], record => {
6     // Create an object to hold name data for the contact
7     const nameData = {
8       firstname: 'John',
9       middlename: 'Doe',
10      lastname: 'Smith'
11    };
12
13    // Create a contact record using the promise method
14    let createRecordPromise = record.create.promise({
15      type: record.Type.CONTACT,
16      isDynamic: true
17    });
18
19    // When the promise is fulfilled, set the values of the subsidiary,
20    // firstname, middlename, and lastname properties, and save the
21    // record
22    createRecordPromise.then(objRecord => {
23      log.debug('Start evaluating promise content...');
24      objRecord.setValue({
25        fieldId: 'subsidiary',
26        value: '1'
27      });
28      for (let key in nameData) {
29        if (nameData.hasOwnProperty(key)) {

```

```

30         objRecord.setValue({
31             fieldId: key,
32             value: nameData[key]
33         });
34     }
35 }
36 let recordId = objRecord.save({
37     enableSourcing: false,
38     ignoreMandatoryFields: false
39 });
40 }, function(e) {
41     log.error('Unable to create contact', e.name);
42 });
43 });

```

Create Multiple Sales Records Using a Scheduled Script

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a scheduled script to create multiple sales records and log the record creation progress.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType ScheduledScript
4   */
5
6  // This script creates multiple sales records and logs the record creation progress.
7  define(['N/runtime', 'N/record'], function(runtime, record) {
8      return {
9          execute: function(context) {
10             var script = runtime.getCurrentScript();
11             for (x = 0; x < 500; x++) {
12                 var rec = record.create({
13                     type: record.Type.SALES_ORDER
14                 });
15                 script.percentComplete = (x * 100)/500;
16                 log.debug({
17                     title: 'New Sales Orders',
18                     details: 'Record creation progress: ' + script.percentComplete + '%'
19                 });
20             }
21         }
22     };
23 });

```

N/recordContext Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/recordContext Module](#):

- [Get the Localization Context of an Employee Record](#)

Get the Localization Context of an Employee Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to retrieve the localization context value for a record. The sample uses two different approaches depending on whether the record is loaded in the script. The subsidiary values are hard-coded. If you run this script in your account, be sure to replace the hard-coded values with valid values from your account.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topic [SuiteScript 2.x Script Basics](#) SuiteScript 2.0 Script Basics and [SuiteScript 2.x Script Types](#) SuiteScript 2.0 Script Types.

```

1  /**
2   *@NApiVersion 2.x
3   */
4   // This example calls the getContext() API with options recordType, recordId and record.
5   require(['N/record', 'N/recordContext'],
6     function(record, recordContext) {
7       // Create record
8       var employee = record.create({
9         type : record.Type.EMPLOYEE,
10        isDynamic: true
11      })
12      // Setup CA subsidiary
13      employee.setValue('subsidiary', 2);
14      employee.setValue('entityid', 'test_emp_' + Date.now())
15      employeeId = employee.save()
16
17      // getContext() with options recordType, recordId and contextTypes
18      var employeeContext = recordContext.getContext({
19        recordType : record.Type.EMPLOYEE,
20        recordId: employeeId,
21        contextTypes: [recordContext.ContextType.LOCALIZATION]
22      })
23      log.debug(employeeContext);
24      // expected log will list {"localization":["CA"]}
25      // Change employee subsidiary to AU
26      employee.setValue('subsidiary', 3);
27      // getContext() with options record and contextTypes
28      var employeeContext = recordContext.getContext({
29        record : employee,
30        contextTypes: [recordContext.ContextType.LOCALIZATION]
31      })
32      log.debug(employeeContext);
33      // expected log will list {"localization":["AU"]}
34      // Delete record
35      record.delete({
36        type : record.Type.EMPLOYEE,
37        id: 1
38      })
39    });

```

N/redirect Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/redirect Module](#):

- [Set a Redirect URL to a Newly Created Task Record](#)

Set a Redirect URL to a Newly Created Task Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample sets the redirect URL to a newly created task record. To set a redirect using a record ID, the record must have been previously submitted to NetSuite.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/record', 'N/redirect'], function(record, redirect) {
6      function redirectToTaskRecord() {
7          var taskTitle = 'New Opportunity';
8          var taskRecord = record.create({
9              type: record.Type.TASK
10         });
11         taskRecord.setValue('title', taskTitle);
12
13         var taskRecordId = taskRecord.save();
14
15         redirect.toRecord({
16             type: record.Type.TASK,
17             id: taskRecordId
18         });
19     }
20
21     redirectToTaskRecord();
22 });

```

N/render Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/render Module](#):

- [Generate a PDF from a Raw XML String](#)
- [Render a PDF](#)
- [Render a Transaction Record Into an HTML Page](#)
- [Render an Invoice Into a PDF Using an XML Template](#)
- [Render Search Results Into a PDF](#)

Generate a PDF from a Raw XML String

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to generate a PDF from a raw XML string.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/render'],
6    function(render) {
7      function generatePdfFileFromRawXml() {
8          var xmlStr = "<?xml version='1.0'?'>\n" +
9            "<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\">\n" +
10           "<pdf>\n<body font-size='18'>\nHello World!\n</body>\n</pdf>";
11          var pdfFile = render.xmlToPdf({
12              xmlString: xmlStr
13          });
14      }
15      generatePdfFileFromRawXml();
16  });

```

Render a PDF

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to render a PDF.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/xml'], function(xml) {
6      return {
7          onRequest: function(context) {
8              var xml = "<?xml version='1.0' encoding='UTF-8'?'>\n" +
9                "<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\">\n" +
10               "<pdf lang='ru-RU' xml:lang='ru-RU'>\n" +
11               "<head>\n" +
12               "<link name='russianfont' type='font' subtype='opentype' " + "src='\"NetSuiteFonts/verdana.ttf\"' " + "src-
bold='\"NetSuiteFonts/verdanab.ttf\"' " + "src-italic='\"NetSuiteFonts/verdanai.ttf\"' " + "src-bolditalic='\"NetSuiteFonts/verdan
abi.ttf\"' " + "bytes='2'/'>\n" +
13               "</head>\n" +
14               "<body font-family='russianfont' font-size='18'>\nРусский текст</body>\n" +
15               "</pdf>";
16              context.response.renderPdf(xml);
17          }
18      }
19  });

```

Render a Transaction Record Into an HTML Page

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to render a transaction record into an HTML page.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Note: The `entityId` value in this sample is a placeholder. Before using this sample, replace the placeholder values with valid values from your NetSuite account.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/render'],
6    function(render) {
7      function renderTransactionToHtml() {
8          var transactionFile = render.transaction({
9              entityId: 23,
10             printMode: render.PrintMode.HTML
11         });
12     }
13     renderTransactionToHtml();
14 });

```

Render an Invoice Into a PDF Using an XML Template

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to render an invoice into a PDF using an XML template in the File Cabinet. This sample requires the Advanced PDF/HTML Templates feature.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  // This sample shows how to render an invoice into a PDF file using an XML template in the file cabinet.
6  // Note that this example requires the Advanced PDF/HTML Templates feature.
7  require(['N/render', 'N/file', 'N/record'],
8    function(render, file, record) {
9      function renderRecordToPdfWithTemplate() {
10         var xmlTemplateFile = file.load('Templates/PDF Templates/invoicePDFTemplate.xml');
11         var renderer = render.create();
12         renderer.templateContent = xmlTemplateFile.getContents();
13         renderer.addRecord('grecord', record.load({
14             type: record.Type.INVOICE,
15             id: 37
16         }));
17         var invoicePdf = renderer.renderAsPdf();
18     }
19     renderRecordToPdfWithTemplate();
20 });

```

In the preceding sample, the `invoicePDFTemplate.xml` file was referenced in the File Cabinet. This file is similar to the Standard Invoice PDF/HTML Template found in Customization > Forms > Advanced PDF/HTML Templates.

Render Search Results Into a PDF

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to render search results into a PDF.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5   // This sample shows how to render search results into a PDF file.
6   define(['N/render', 'N/search'], function(render, search) {
7       function onRequest(options) {
8           var request = options.request;
9           var response = options.response;
10
11           var xmlStr = '<?xml version="1.0" encoding="UTF-8"?>\n' +
12               '<!DOCTYPE pdf PUBLIC "-//big.faceless.org//report" "report-1.1.dtd">\n' +
13               '<pdf lang="ru-RU" xml:lang="ru-RU">\n' + "<head>\n" +
14               '<link name="russianfont" type="font" subtype="opentype" ' + 'src="NetSuiteFonts/verdana.ttf" ' + "src-
15               bold="NetSuiteFonts/verdanab.ttf" ' + 'src-italic="NetSuiteFonts/verdanai.ttf" ' + "src-bolditalic="NetSuiteFonts/verdan
16               abi.ttf" ' + 'bytes="2"/>\n' + "</head>\n" +
17               '<body font-family="russianfont" font-size="18">\n????? ????</body>\n' + "</pdf>";
18
19           var rs = search.create({
20               type: search.Type.TRANSACTION,
21               columns: ['trandate', 'amount', 'entity'],
22               filters: []
23           }).run();
24
25           var results = rs.getRange(0, 1000);
26           var renderer = render.create();
27           renderer.templateContent = xmlStr;
28           renderer.addSearchResults({
29               templateName: 'exampleName',
30               searchResult: results
31           });
32
33           var newfile = renderer.renderAsPdf();
34           response.writeFile(newfile, false);
35       }
36
37       return {
38           onRequest: onRequest
39       };
40   });

```

N/runtime Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/runtime Module](#):

- [Create Multiple Sales Records Using a Scheduled Script](#)
- [Return User and Session Information](#)

Create Multiple Sales Records Using a Scheduled Script

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a scheduled script to create multiple sales records and log the record creation progress.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType ScheduledScript
4   */
5
6   // This script creates multiple sales records and logs the record creation progress.
7   define(['N/runtime', 'N/record'], function(runtime, record) {
8       return {
9           execute: function(context) {
10              var script = runtime.getCurrentScript();
11              for (x = 0; x < 500; x++) {
12                  var rec = record.create({
13                      type: record.Type.SALES_ORDER
14                  });
15                  script.percentComplete = (x * 100)/500;
16                  log.debug({
17                      title: 'New Sales Orders',
18                      details: 'Record creation progress: ' + script.percentComplete + '%'
19                  });
20              }
21          }
22      };
23  });

```

Return User and Session Information

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use a Suitelet to write user and session information for the currently executing script to the response.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5
6   // This script writes user and session information for the currently executing script to the response.
7   define(['N/runtime'], function(runtime) {
8       function onRequest(context) {
9           var remainingUsage = runtime.getCurrentScript().getRemainingUsage();
10          var userRole = runtime.getCurrentUser().role;

```

```

11     var currentSession = runtime.getCurrentSession();
12
13     // Set the current sessions's scope
14     currentSession.set({
15         name: 'scope',
16         value: 'global'
17     });
18
19     var sessionScope = runtime.getCurrentSession().get({
20         name: 'scope'
21     });
22
23     log.debug('Remaining Usage:', remainingUsage);
24     log.debug('Role:', userRole);
25     log.debug('Session Scope:', sessionScope);
26
27     context.response.write('Executing under role: ' + userRole
28         + '. Session scope: ' + sessionScope + '.');
29 }
30 return {
31     onRequest: onRequest
32 };
33 });

```

N/search Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/search Module](#):

- [Create a Search for a Custom Record Type](#)
- [Delete a Saved Search](#)
- [Load a Search for Sales Order Records and Return the First 100 Search Results](#)
- [Load a Search for Sales Order Records and Use a Callback Function to Process Results](#)
- [Load and Run a Paginated Search and Process the Results](#)
- [Search for Customer Records and Log First 50 Results](#)
- [Search for Items in a Custom List](#)
- [Search for Sales Order Records](#)

Create a Search for a Custom Record Type

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

The following sample creates a search for a custom record type. To search for a custom record type, you must specify a type of search `.Type.CUSTOM_RECORD` and add the ID of the custom record type (as a string). In this sample, the ID of the custom record type is 6. The custom record also includes a custom field named `custrecord1`.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      var myCustomRecordSearch = search.create({
7          type: search.Type.CUSTOM_RECORD + '6';
8          title: 'My Search Title',
9          columns: ['custrecord1']
10     }).run().each(function(result) {
11         // Process each result
12         return true;
13     });
14 });

```

Delete a Saved Search

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to delete a saved search.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      function deleteSearch() {
7          search.delete({
8              id: 'customsearch_my_so_search'
9          });
10     }
11
12     deleteSearch();
13 });

```

Load a Search for Sales Order Records and Return the First 100 Search Results

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads and runs a saved search for sales order records. The sample obtains the first 100 rows of search results.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**

```

```

2  /**
3   * @NApiVersion 2.x
4   */
5  require(['N/search'], function(search) {
6      function runSearchAndFetchResult() {
7          var mySearch = search.load({
8              id: 'customsearch_my_so_search'
9          });
10
11          var searchResult = mySearch.run().getRange({
12              start: 0,
13              end: 100
14          });
15          for (var i = 0; i < searchResult.length; i++) {
16              var entity = searchResult[i].getValue({
17                  name: 'entity'
18              });
19              var subsidiary = searchResult[i].getValue({
20                  name: 'subsidiary'
21              });
22          }
23      }
24
25      runSearchAndFetchResult();
26  });

```

Load a Search for Sales Order Records and Use a Callback Function to Process Results

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads and runs a saved search for sales order records. The sample uses the each callback function to process the results.

i Note: This sample script uses the require function so that you can copy it into the SuiteScript Debugger and test it. You must use the define function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      function loadAndRunSearch() {
7          var mySearch = search.load({
8              id: 'customsearch_my_so_search'
9          });
10
11          mySearch.run().each(function(result) {
12              var entity = result.getValue({
13                  name: 'entity'
14              });
15              var subsidiary = result.getValue({
16                  name: 'subsidiary'
17              });
18
19              return true;
20          });
21      }
22
23      loadAndRunSearch();
24  });

```


Load and Run a Paginated Search and Process the Results

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads and runs a saved search for sales order records. The sample uses the `forEach` callback function to process the paginated results.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      function loadAndRunSearch() {
7          var mySearch = search.load({
8              id: 'customsearch_my_so_search'
9          });
10
11         var myPagedData = mySearch.runPaged();
12         myPagedData.pageRanges.forEach(function(pageRange){
13             var myPage = myPagedData.fetch({index: pageRange.index});
14             myPage.data.forEach(function(result){
15                 var entity = result.getValue({
16                     name: 'entity'
17                 });
18                 var subsidiary = result.getValue({
19                     name: 'subsidiary'
20                 });
21             });
22         });
23     }
24
25     loadAndRunSearch();
26 });

```

Search for Customer Records and Log First 50 Results

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a search for customer records. The sample specifies several result columns and one filter, and it logs the first 50 search results.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {

```

```

6   var mySearch = search.create({
7       type: search.Type.CUSTOMER,
8       columns: ['entityid', 'firstname', 'lastname', 'salesrep'],
9       filters: ['entityid', 'contains', 'Adam']
10  });
11
12  var myResultSet = mySearch.run();
13
14  var resultRange = myResultSet.getRange({
15      start: 0,
16      end: 50
17  });
18
19  for (var i = 0; i < resultRange.length; i++) {
20      log.debug(resultRange[i]);
21  }
22  });

```

Search for Items in a Custom List

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a search for items in a custom list. It searches for the internal ID value of an abbreviation in a custom list named `customlist_mylist`.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      var internalId = -1;
7      var myCustomListSearch = search.create({
8          type: 'customlist_mylist',
9          columns: [
10             { name: 'internalId' },
11             { name: 'abbreviation' }
12         ]
13     });
14
15     myCustomListSearch.filters = [
16         search.createFilter({
17             name: 'formulatext',
18             formula: '{abbreviation}',
19             operator: search.Operator.IS,
20             values: abbreviation
21         });
22     ]
23
24     var resultSet = myCustomListSearch.run();
25     var results = resultSet.getRange({
26         start: 0,
27         end: 1
28     });
29     for(var i in results) {
30         // log.debug('Found custom list record', results[i]);
31         internalId = results[i].getValue({
32             name: 'internalId'
33         });
34     };

```

```
35 | });
```

Search for Sales Order Records

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a search for sales order records and saves it. The sample specifies several result columns and two filters.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```
1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/search'], function(search) {
6      function createSearch() {
7          var mySalesOrderSearch = search.create({
8              type: search.Type.SALES_ORDER,
9              title: 'My SalesOrder Search',
10             id: 'customsearch_my_so_search',
11             columns: ['entity', 'subsidiary', 'name', 'currency'],
12             filters: [
13                 ['mainline', 'is', 'T'],
14                 'and', ['subsidiary.name', 'contains', 'CAD']
15             ]
16         });
17     }
18     mySalesOrderSearch.save();
19 }
20
21 createSearch();
22 });
```

N/sftp Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/sftp Module](#):

- [Manage Files and Directories](#)
- [Set Conditional Default Settings Using N/sftp Enums](#)
- [Upload and Download a File](#)

Manage Files and Directories

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how you can manage files and directories.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/ftp', 'N/file'], (ftp, file) => {
5       // Establish a connection
6       log.debug('Establishing SFTP connection...');
7       let connection = ftp.createConnection({
8           username: 'sftpuser',
9           keyId: 'custkeysftp_nft_demo_key',
10          url: 'myurl',
11          port: 22,
12          directory: 'inbound',
13          hostKey: 'myhostkey'
14      });
15      log.debug('Connection established!');
16
17      // List the directory and log the number of elements there
18      let list = connection.list({
19          path: 'yyy/test'
20      });
21      log.debug('Items in directory "test" at the beginning: ' + list.length);
22
23      // Generate the test file
24      log.debug('Generating test file...');
25      let myFileToUpload = file.create({
26          name: 'asdf.txt',
27          fileType: file.Type.PLAINTEXT,
28          contents: 'I am a test file.'
29      });
30      log.debug('Test file generated, uploading to "test" directory...');
31
32      // Upload the test file
33      connection.upload({
34          directory: 'yyy/test',
35          filename: 'af.txt',
36          file: myFileToUpload,
37          replaceExisting: true
38      });
39      log.debug('Upload complete!');
40
41      // List the directory to confirm there is one more file than before
42      list = connection.list({
43          path: 'yyy/test'
44      });
45      log.debug('Items in directory "test" after the upload: ' + list.length);
46
47      // Create a new directory
48      log.debug('Creating directory "test2"...');
49      try {
50          connection.makeDirectory({
51              path: 'yyy/test2'
52          });
53          log.debug('Directory created. ');
54      } catch (e) {
55          log.debug('Directory not created. ');
56          log.error(e.message);
57      }
58      list = connection.list({
59          path: 'yyy/test2'
60      });
61      log.debug('Items in directory "test2": ' + list.length);

```

```

62
63 // Move the test file to the new directory
64 log.debug('Moving the test file from "test" to "test2"...');
65 connection.move({
66     from: 'yyy/test/af.txt',
67     to: 'yyy/test2/af.txt'
68 });
69 log.debug('File moved!');
70
71 // List the original directory again to see the file is moved
72 list = connection.list({
73     path: 'yyy/test'
74 });
75 log.debug('Items in directory "test" after the upload: ' + list.length);
76
77 // List the new directory for the file
78 list = connection.list({path: 'yyy/test2'});
79 log.debug('Items in directory "test2" after the upload: ' + list.length);
80 log.debug(JSON.stringify(list));
81
82 // Try to remove the directory
83 log.debug('Removing directory "test2"...');
84 try {
85     connection.removeDirectory({
86         path: 'yyy/test2'
87     });
88     log.debug('Directory removed!');
89 } catch (e) {
90     log.debug('Directory not removed!');
91     log.error(e.message);
92 }
93
94 // The directory is not empty so delete the file first
95 log.debug('Removing test file from "test2" directory...');
96 connection.removeFile({
97     path: 'yyy/test2/af.txt'
98 });
99 log.debug('Test file removed!');
100
101 list = connection.list({
102     path: 'yyy/test2'
103 });
104 log.debug('Items in directory "test2": ' + list.length);
105
106 // Try to remove the directory again
107 log.debug('Removing directory "test2"...');
108 try {
109     connection.removeDirectory({
110         path: 'yyy/test2'
111     });
112     log.debug('Directory removed!');
113 } catch (e) {
114     log.debug('Directory not removed!');
115     log.error(e.message);
116 }
117
118 // Try to list the removed directory
119 log.debug('Trying to list directory "test2"...');
120 try {
121     list = connection.list({
122         path: 'yyy/test2'
123     });
124 } catch (e) {
125     log.error(e.message);
126 }
127 });

```

Set Conditional Default Settings Using N/sftp Enums

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use the N/sftp module enums to set conditional default settings. The sample creates a secure connection and attempts to upload and add a large file.

Note: This script sample uses the define function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the require function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/file', 'N/sftp', 'N/error'],
8  function(file, sftp, error) {
9      return {
10         beforeLoad: function(){
11             var portNumber = -1;
12             var connectTimeout = -1;
13             var transferTimeout = -1;
14             //these variables can be taken as parameters of the script instead
15
16             if (portNumber < sftp.MIN_PORT_NUMBER || portNumber > sftp.MAX_PORT_NUMBER)
17                 portNumber = sftp.DEFAULT_PORT_NUMBER;
18             if (connectTimeout < sftp.MIN_CONNECT_TIMEOUT) connectTimeout = sftp.MIN_CONNECT_TIMEOUT; else if (connectTimeout >
19 sftp.MAX_CONNECT_TIMEOUT)
20                 connectTimeout = sftp.MAX_CONNECT_TIMEOUT;
21
22             var connection = sftp.createConnection({
23                 username: 'sftpuser',
24                 keyId: 'custkey1',
25                 url: '192.168.0.100',
26                 port: portNumber,
27                 directory: 'inbound',
28                 timeout: connectTimeout,
29                 hostKey: "AAAAB3NzaC1yc2EAAAADAQABAAQDMiFkH2vTxditye8nem7+1S3x7dTQR/A67KdsR/5C2wUcDipBzYhHb
nG6Am12Nd2t1M01LnaBZA6/8P4Y9x/sGTxtsdE/MzeGDUBn6HB1QvgIrhX62wgoKGQ+P21EA01+Vz8y3/MB1NmD7Fc62cJ9Mu88YA6jwJOIPZeHYNVyIm90rY6VyzYyvSJh
H0x75XyvGnijJQF4G8C4c8u/UVpF/sE16xKZtly2Rx0aDL2FsDRtpyPmM602/R6ISbsmgab3MzzAEIu+zLDMdIBJn3cDhNt1F7Rar6Tu0u18KCKk8GPxbnxDuG4sCN0nX
PYkDXSMUbm/ocRjYGtqdZUMmeTf3"
30             });
31
32             // can also be a big file (created for example by async search)
33             var myFileToUpload = file.create({
34                 name: 'originalname.txt',
35                 fileType: file.Type.PLAINTEXT,
36                 contents: 'I am a test file.'
37             });
38
39             if (myFileToUpload.size > connection.MAX_FILE_SIZE)
40                 throw error.create({name: "FILE_IS_TOO_BIG", message: "The file you are trying to upload is too big"});
41
42             var minTransferTimeout = 10;
43             if (transferTimeout > connection.MAX_TRANSFER_TIMEOUT)
44                 transferTimeout = connection.MAX_TRANSFER_TIMEOUT;
45             else if (transferTimeout < minTransferTimeout)
46                 transferTimeout = minTransferTimeout;
47
48             connection.upload({
49                 directory: 'files',
50                 filename: 'test.txt',
51                 file: myFileToUpload,
52                 replaceExisting: true,

```

```

52         timeout: transferTimeout
53     });
54 }
55 };
56 });

```

Upload and Download a File

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to upload and download a file.

To obtain a real host key, use `ssh-keyscan <domain>`.

To create a real password GUID, obtain a password value from a credential field on a form. For more information, see the help topic [Form.addCredentialField\(options\)](#). Also see the help topic [N/https Module Script Samples](#) for a Suitelet sample that shows creating a form field that generates a GUID.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/sftp', 'N/file'], (sftp, file) => {
5       const myPwdGuid = "B34672495064525E5D65032D63852301";
6       const myHostKey = "AAA1234567890Q=";
7
8       // Establish a connection to a remote FTP server
9       let connection = sftp.createConnection({
10         username: 'myuser',
11         passwordGuid: myPwdGuid,
12         url: 'host.somewhere.com',
13         directory: 'myuser/wheres/my/file',
14         hostKey: myHostKey
15     });
16
17     // Create a file to upload using the N/file module
18     let myFileToUpload = file.create({
19         name: 'originalname.js',
20         fileType: file.Type.PLAINTEXT,
21         contents: 'I am a test file.'
22     });
23
24     // Upload the file to the remote server
25     connection.upload({
26         directory: 'relative/path/to/remote/dir',
27         filename: 'newFileNameOnServer.js',
28         file: myFileToUpload,
29         replaceExisting: true
30     });
31
32     // Download the file from the remote server
33     let downloadedFile = connection.download({
34         directory: 'relative/path/to/file',
35         filename: 'downloadMe.js'
36     });

```

```
37 | });
```

N/sso Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/sso Module](#):

- [Generate a New OAuth Token](#)
- [Generate a suiteSignOn Token in a Portlet](#)
- [Generate a suiteSignOn Token Using a Suitelet](#)
- [Generate a suiteSignOn Token Using a User Event Script](#)

Generate a New OAuth Token

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to generate a new OAuth token for a user. This sample requires the SuiteSignOn feature.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: The value used in this sample for the `suiteSignOnRecordId` field is a placeholder. Before using this sample, replace the `suiteSignOnRecordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur. Additionally, the SuiteSignOn record you reference must be associated with a specific script. You make this association in the SuiteSignOn record's Connection Points sublist. For help with SuiteSignOn records, see the help topic [Creating SuiteSignOn Records](#).

```
1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script generates a new OAuth token for a user.
6   require(['N/sso'], function(sso) {
7       function generateSSOToken() {
8           var suiteSignOnRecordId = 1;
9           var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
10        }
11        generateSSOToken();
12    });
```

Generate a suiteSignOn Token in a Portlet

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use `generateSuiteSignOnToken(options)` in a portlet script.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: The value used in this sample for the `suiteSignOnRecordId` field is a placeholder. Before using this sample, replace the `suiteSignOnRecordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur. Additionally, the SuiteSignOn record you reference must be associated with a specific script. You make this association in the SuiteSignOn record's Connection Points sublist. For help with SuiteSignOn records, see the help topic [Creating SuiteSignOn Records](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Portlet
4   * @NScriptPortletType form
5   */
6
7  // This script uses generateSuiteSignOnToken in a portlet.
8  define(['N/sso'], function (sso) {
9      function render(context) {
10         var suiteSignOnRecordId = 'customsso_test';
11         var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
12         log.debug(url);
13     }
14     return {
15         render: render
16     };
17 });

```

Generate a suiteSignOn Token Using a Suitelet

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use `generateSuiteSignOnToken(options)` in a Suitelet script.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: The value used in this sample for the `suiteSignOnRecordId` field is a placeholder. Before using this sample, replace the `suiteSignOnRecordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur. Additionally, the SuiteSignOn record you reference must be associated with a specific script. You make this association in the SuiteSignOn record's Connection Points sublist. For help with SuiteSignOn records, see the help topic [Creating SuiteSignOn Records](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5
6  // This script uses generateSuiteSignOnToken in a Suitelet.

```

```

7 define(['N/sso'], function(sso) {
8   function onRequest(context) {
9     var suiteSignOnRecordId = 'customsso_test'; //Replace placeholder values
10    var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
11    log.debug(url);
12  }
13  return {
14    onRequest: onRequest
15  };
16 });

```

Generate a suiteSignOn Token Using a User Event Script

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to use `generateSuiteSignOnToken(options)` in a user event script.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Important: The value used in this sample for the `suiteSignOnRecordId` field is a placeholder. Before using this sample, replace the `suiteSignOnRecordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur. Additionally, the SuiteSignOn record you reference must be associated with a specific script. You make this association in the SuiteSignOn record's Connection Points sublist. For help with SuiteSignOn records, see the help topic [Creating SuiteSignOn Records](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  // This script uses generateSuiteSignOnToken in a user event script.
8  define(['N/sso'], function(sso) {
9    function beforeLoad(context) {
10     var suiteSignOnRecordId = 'customsso_test';
11     var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
12     log.debug(url);
13   }
14   return {
15     beforeLoad: beforeLoad
16   };
17 });

```

N/suiteAppInfo Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/suiteAppInfo Module](#):

- [Retrieve Information for a SuiteApp](#)

Retrieve Information for a SuiteApp

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following script sample performs these tasks:

- Checks to see if a specific bundle is installed
- Checks to see if a specific SDF SuiteApp is installed
- Retrieves a list of successfully installed bundles
- Retrieves a list of successfully installed SDF SuiteApps
- Retrieves a list of bundles in the current account that include a specific script
- Retrieves a list of SDF SuiteApps in the current account that include a specific script

Some of the values in this sample are placeholders, such as the `bundleId` and `suiteAppId` values. Before using this sample, replace all hard-coded values, including IDs, with valid values from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /*
2   * @NApiVersion 2.x
3   *
4   */
5
6  // This script sample uses each method available in the N/suiteAppInfo module to retrieve information about installed bundles and SuiteApps installed in
   the current account.
7
8  require(['N/suiteAppInfo'], function (suiteAppInfo){
9      var isBundleInstalled = suiteAppInfo.isBundleInstalled({
10         bundleId: '1234'
11     });
12
13     var isSuiteAppInstalled = suiteAppInfo.isSuiteAppInstalled({
14         suiteAppId: '789'
15     });
16
17     var allMyBundlesInstalled = suiteAppInfo.listInstalledBundles();
18
19     var allMySuiteAppsInstalled = suiteAppInfo.listInstalledSuiteApps();
20
21     var myScripts = ["scriptA", "scriptB", "scriptC"];
22     var scriptInBundles = suiteAppInfo.listBundlesContainingScripts({
23         scriptIds: myScripts
24     });
25
26     var scriptInSuiteApps = suiteAppInfo.listSuiteAppsContainingScripts({
27         scriptsIds: myScripts
28     });
29 })

```

N/task Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/task Module](#):

- [Create and Submit a Map/Reduce Script Task](#)
- [Create and Submit a Task with Dependent Scripts](#)
- [Create and Submit an Asynchronous Search Task and Export the Results into a CSV File](#)
- [Submit a Record Action Task and Check Status](#)

Create and Submit a Map/Reduce Script Task

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample submits a map/reduce script task for processing. Before you use this sample, you must create a map/reduce script file, upload the file to your NetSuite account, and create a script record and script deployment record for it. For help working with map/reduce scripts, see the help topic [SuiteScript 2.x Map/Reduce Script Type](#). You must also edit the sample and replace all hard-coded IDs with values that are valid in your NetSuite account.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/task', 'N/runtime', 'N/email'], (task, runtime, email) => {
5       // Store the script ID of the script to submit
6       //
7       // Update the following statement so it uses the script ID
8       // of the map/reduce script record you want to submit
9       const mapReduceScriptId = 'customscript_test_mapreduce_script';
10
11      // Create a map/reduce task
12      //
13      // Update the deploymentId parameter to use the script ID of
14      // the deployment record for your map/reduce script
15      let mrTask = task.create({
16          taskType: task.TaskType.MAP_REDUCE,
17          scriptId: mapReduceScriptId,
18          deploymentId: 'customdeploy_test_mapreduce_script'
19      });
20
21      // Submit the map/reduce task
22      let mrTaskId = mrTask.submit();
23
24      // Check the status of the task, and send an email if the
25      // task has a status of FAILED
26      //
27      // Update the authorId value with the internal ID of the user
28      // who is the email sender. Update the recipientEmail value
29      // with the email address of the recipient.
30      let taskStatus = task.checkStatus(mrTaskId);
31      if (taskStatus.status === 'FAILED') {
32          const authorId = -5;
33          const recipientEmail = 'notify@myCompany.com';
34          email.send({

```

```

35     author: authorId,
36     recipients: recipientEmail,
37     subject: 'Failure executing map/reduce job!',
38     body: 'Map reduce task: ' + mapReduceScriptId + ' has failed.'
39   });
40 }
41 });

```

Create and Submit a Task with Dependent Scripts

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a scheduled script task and a map/reduce script task. It then creates an asynchronous search task and adds the scheduled script task and the map/reduce script task to the search task as dependent scripts. These scripts are processed when the search task is complete. For more information, see the help topic [SuiteCloud Processors](#).

This sample refers to two script parameters: `custscript_ss_as_srch_res` for the scheduled script, and `custscript_mr_as_srch_res` for the map/reduce script. These parameters are used to pass the location of the CSV file to the dependent scripts, which is shown in the second and third code samples below. Before using this sample, create these parameters in the script record. For more information, see the help topic [Creating Script Parameters](#).

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/task'], function(task) {
6    // Specify a file for the search results
7    var asyncSearchResultFile = 'SuiteScripts/ExportFile.csv';
8
9    // Create a scheduled script task
10   var scheduledScript = task.create({
11     taskType: task.TaskType.SCHEDULED_SCRIPT
12   });
13   scheduledScript.scriptId = 'customscript_as_ftr_ss';
14   scheduledScript.deploymentId = 'customdeploy_ss_dpl';
15   scheduledScript.params = {
16     'custscript_ss_as_srch_res' : asyncSearchResultFile
17   };
18
19   // Create a map/reduce script task
20   var mapReduceScript = task.create({
21     taskType: task.TaskType.MAP_REDUCE
22   });
23   mapReduceScript.scriptId = 'customscript_as_ftr_mr';
24   mapReduceScript.deploymentId = 'customdeploy_mr_dpl';
25   mapReduceScript.params = {
26     'custscript_mr_as_srch_res' : asyncSearchResultFile
27   };
28
29   // Create the search task
30   var asyncTask = task.create({
31     taskType: task.TaskType.SEARCH
32   });
33   asyncTask.savedSearchId = 'customsearch35';

```

```

34     asyncTask.filePath = asyncSearchResultFile;
35
36     // Add dependent scripts to the search task before it is submitted
37     asyncTask.addInboundDependency(scheduledScript);
38     asyncTask.addInboundDependency(mapReduceScript);
39
40     // Submit the search task
41     var asyncTaskId = asyncTask.submit();
42 });

```

To read the contents of the search results file in a dependent scheduled script, consider the following script sample:

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType ScheduledScript
4   */
5  define(['N/file', 'N/log', 'N/email', 'N/runtime'], function(file, log, email, runtime) {
6      // Load the search results file and send an email with the file attached and
7      // the number of rows in the file
8
9      function execute(context) {
10         // Read a CSV file and return the number of rows minus the header row
11         function numberOfRows(csvFileId) {
12             var invoiceFile = file.load({
13                 id: csvFileId
14             });
15             var iterator = invoiceFile.lines.iterator();
16             var noOfLines = 0;
17
18             // Skip the first row (the header row)
19             iterator.each(function() {
20                 return false;
21             });
22
23             // Process the rest of the rows
24             iterator.each(function() {
25                 noOfLines++;
26                 return true;
27             });
28
29             return noOfLines;
30         }
31
32         // Send an email to the user who ran the script, and attach the
33         // CSV file with the search results
34         function sendEmailWithAttachment(csvFileId) {
35             var noOfRows = numberOfRows(csvFileId);
36             var userId = runtime.getCurrentUser().id;
37             var fileObj = file.load({
38                 id: csvFileId
39             });
40
41             email.send({
42                 author: userId,
43                 recipients: userId,
44                 subject: 'Search completed',
45                 body: 'CSV file attached, ' + noOfRows + ' record(s) found.',
46                 attachments: [fileObj]
47             });
48         }
49
50         // Retrieve the ID of the search results file
51         //
52         // Update the name parameter to use the script ID of the original
53         // search task
54         var resFileId = runtime.getCurrentScript().getParameter({
55             name: 'custscript_ss_as_srch_res'
56         });
57
58         if (!resFileId) {
59             log.error('Could not obtain file content from the specified ID.');
```

```

60     return;
61 }
62
63 log.debug({
64     title: 'search - numberOfRows',
65     details: numberOfRows(resFileId)
66 });
67 sendEmailWithAttachment(resFileId);
68 }
69
70 return {
71     execute: execute
72 };
73 });

```

To read the contents of the search results file in a dependent map/reduce script, consider the following script sample:

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType MapReduceScript
4   * @NModuleScope SameAccount
5   */
6  define(['N/runtime', 'N/file', 'N/log', 'N/email'], function(runtime, file, log, email) {
7      // Load the search results file, count the number of letters in the file, and
8      // store this count in another file
9
10     function getInputData() {
11         // Retrieve the ID of the search results file
12         //
13         // Update the completionScriptParameterName value to use the script
14         // ID of the original search task
15         var completionScriptParameterName = 'custscript_mr_as_srch_res';
16         var resFileId = runtime.getCurrentScript().getParameter({
17             name: completionScriptParameterName
18         });
19
20         if (!resFileId) {
21             log.error({
22                 details: 'resFileId is not valid. Please check the script parameter stored in the completionScriptParameterName
23                 variable in getInputData().'
24             });
25         }
26
27         return {
28             type: 'file',
29             id: resFileId
30         };
31     }
32
33     function map(context) {
34         var email = context.value.split(',')[1];
35         if ("Email" !== email) {
36             var splitEmail = email.split('@');
37             context.write(splitEmail[splitEmail.length-1], 1);
38         }
39     }
40
41     function reduce(context) {
42         context.write(context.key, context.values.length);
43     }
44
45     function summarize(summary) {
46         var type = summary.toString();
47         log.audit({title: type + ' Usage Consumed ', details: summary.usage});
48         log.audit({title: type + ' Concurrency Number ', details: summary.concurrency});
49         log.audit({title: type + ' Number of Yields ', details: summary.yields});
50
51         var contents = '';
52         summary.output.iterator().each(function(key, value) {
53             contents += (key + ' ' + value + '\n');
54         });
55         return true;
56     }
57 }

```

```

54     });
55
56     // Create the output file
57     //
58     // Update the name parameter to use the file name of the output file
59     var fileObj = file.create({
60         name: 'domainCount.txt',
61         fileType: file.Type.PLAINTEXT,
62         contents: contents
63     });
64
65     // Specify the folder location of the output file, and save the file
66     //
67     // Update the fileObj.folder property with the ID of the folder in
68     // the file cabinet that contains the output file
69     fileObj.folder = -15;
70     fileObj.save();
71 }
72
73 return {
74     getInputData: getInputData,
75     map: map,
76     reduce: reduce,
77     summarize: summarize
78 };
79 });

```

Create and Submit an Asynchronous Search Task and Export the Results into a CSV File

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates an asynchronous search task to execute a saved search and export the results of the search into a CSV file stored in the File Cabinet. After the search task is submitted, the sample retrieves the task status using the task ID. Some of the values in this sample are placeholders. Before using this sample, replace all hard-coded values, such as IDs and file paths, with valid values from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   require(['N/task'], task => {
5       // Do one of the following:
6       //
7       // - Create a saved search and capture its ID. To do this, you can use
8       //   the following code snippet (replacing the type, id, filters, and
9       //   columns values as appropriate):
10      //
11      // let mySearch = search.create({
12      //     type: search.Type.SALES_ORDER,
13      //     id: 'customsearch_my_search';

```



```

14 // filters: [...],
15 // columns: [...]
16 // });
17 // mySearch.save();
18 // let savedSearchId = mySearch.searchId;
19 //
20 // - Use the ID of an existing saved search. This is the approach that
21 // this script sample uses. Update the following statement with the
22 // internal ID of the search you want to use.
23 const savedSearchId = 669;
24
25 // Create the search task
26 let myTask = task.create({
27     taskType: task.TaskType.SEARCH
28 });
29 myTask.savedSearchId = savedSearchId;
30
31 // Specify the ID of the file that search results will be exported into
32 //
33 // Update the following statement so it uses the internal ID of the file
34 // you want to use
35 myTask.fileId = 448;
36
37 // Submit the search task
38 let myTaskId = myTask.submit();
39
40 // Retrieve the status of the search task
41 let taskStatus = task.checkStatus({
42     taskId: myTaskId
43 });
44
45 // Optionally, add logic that executes when the task is complete
46 if (taskStatus.status === task.TaskStatus.COMPLETE) {
47     // Add any code that is appropriate. For example, if this script created
48     // a saved search, you may want to delete it.
49 }
50 });

```

Submit a Record Action Task and Check Status

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to submit a record action task and then check its status. For details about record action tasks, see the help topics [task.RecordActionTask](#) and [task.RecordActionTaskStatus](#).

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1 /**
2  * @NApiVersion 2.x
3  */
4
5 require(['N/task'], function(task) {
6     var recordActionTask = task.create({
7         taskType: task.TaskType.RECORD_ACTION
8     });
9     recordActionTask.recordType = 'timebill';
10    recordActionTask.action = 'approve';
11    recordActionTask.params = [
12        {recordId: 1, note: 'This is a note for 1'},
13        {recordId: 5, note: 'This is a note for 5'},

```

```

14     {recordId: 23, note: 'This is a note for 23'}
15   ];
16
17   var handle = recordActionTask.submit();
18
19   var res = task.checkStatus({
20     taskId: handle
21   }); // Returns a RecordActionTaskStatus object
22   log.debug('Initial status: ' + res.status);
23 });

```

N/task/accounting/recognition Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/task/accounting/recognition Module](#):

- [Merge Revenue Arrangements Using a Saved Search](#)
- [Merge Revenue Arrangements Using an Ad-Hoc Search](#)
- [Merge Revenue Elements Using Internal IDs](#)

Merge Revenue Arrangements Using a Saved Search

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads a saved search for revenue arrangement records. It obtains the value of the `internalid` field from each record in the result set, and it adds the values to an array. It calls [recognition.create\(options\)](#) to create a merge task for revenue arrangement records, uses the array as the list of revenue arrangement records to merge, and submits the merge task. The sample also checks the status of the merge task and logs status information.

If you run this sample code in your account, be sure to use a saved search for valid revenue arrangement records in your account.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/task/accounting/recognition', 'N/search'], function(recognition, search){
6    var mySearch = search.load({
7      id: 'customsearch22'
8    });
9
10   var elementsList = [];
11   mySearch.run().each(function(result) {
12     var id = result.getValue({
13       name: 'internalid'

```

```

14     });
15     elementsList.push(id);
16 });
17
18 var recognitionTask = recognition.create({
19     taskType: recognition.TaskType.MERGE_ARRANGEMENTS_TASK
20 });
21
22 recognitionTask.arrangements = elementsList;
23 recognitionTask.revenueArrangementDate = new Date(2019, 2, 10);
24
25 var taskStatusId = recognitionTask.submit();
26 log.debug('taskId = ' + taskStatusId);
27
28 var mergeTaskState = recognition.checkStatus({
29     taskId: taskStatusId
30 });
31
32 log.debug('Submission ID = ' + mergeTaskState.submissionId);
33 log.debug('Resulting Arrangement ID = ' + mergeTaskState.resultingArrangement);
34 log.debug('status = ' + mergeTaskState.status);
35 log.debug('Error message = ' + mergeTaskState.errorMessage);
36 });

```

Merge Revenue Arrangements Using an Ad-Hoc Search

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates an ad-hoc search for revenue element records. It obtains the first 50 results, obtains the value of the elementsList field from each record in the result set, and adds the values to an array. It calls `recognition.create(options)` to create a merge task for revenue element records, uses the array as the list of revenue elements records to merge, and submits the merge task. This sample also checks the status of the merge task and logs status information.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/task/accounting/recognition', 'N/search'], function(recognition, search) {
6      var elementsList = [];
7      var rs = search.create({
8          type: 'revenueelement',
9          columns: [
10             'internalid'
11         ]
12     }).run();
13
14     var results = rs.getRange(0, 50);
15     for (var i = 0; i < results.length; i++) {
16         var id = result.getValue('elementsList');
17         elementsList.push(id);
18     }
19
20     var t = recognition.create({
21         taskType: recognition.TaskType.MERGE_ELEMENTS_TASK
22     });
23     t.elements = elementsList;

```

```

24 |     t.revenueArrangementDate = new Date(2019, 1, 1);
25 |
26 |     var taskId = t.submit();
27 |     log.debug('Initial status: ' + res.status);
28 | });

```

Merge Revenue Elements Using Internal IDs

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample adds the internal IDs of several revenue element records to an array. It calls [recognition.create\(options\)](#) to create a merge task for revenue element records, uses the array as the list of revenue element records to merge, and submits the merge task. The sample also checks the status of the merge task.

If you run this sample code in your account, be sure to use the internal IDs of valid revenue element records in your account.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1 | /**
2 |  * @NApiVersion 2.x
3 |  */
4 |
5 | require(['N/task/accounting/recognition'], function(recognition){
6 |     var elementsList = [];
7 |     elementsList.push(401);
8 |     elementsList.push(402);
9 |
10 |    var recognitionTask = recognition.create({
11 |        taskType: recognition.TaskType.MERGE_ELEMENTS_TASK
12 |    });
13 |    recognitionTask.elements = elementsList;
14 |    var taskStatusId = recognitionTask.submit();
15 |
16 |    var mergeTaskState = recognition.checkStatus({
17 |        taskId: taskStatusId
18 |    });
19 |
20 |    log.debug('Submission ID = ' + mergeTaskState.submissionId);
21 |    log.debug('Resulting Arrangement ID = ' + mergeTaskState.resultingArrangement);
22 |    log.debug('status = ' + mergeTaskState.status);
23 | });

```

N/transaction Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/transaction Module](#):

- [Void a Sales Order Transaction](#)

Void a Sales Order Transaction

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a sales order record, saves it, then voids the sales order. Before creating the sales order record, the sample loads a set of accounting preferences from the current NetSuite account, specifies that the REVERSALVOIDING preference should be disabled (set to false), and saves the preferences. This sample works only in NetSuite OneWorld accounts. Be sure to replace hard-coded values (such as record IDs) with valid values from your NetSuite account.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/transaction', 'N/config', 'N/record'], function(transaction, config, record) {
6      function voidSalesOrder() {
7          var accountingConfig = config.load({
8              type: config.Type.ACCOUNTING_PREFERENCES
9          });
10         accountingConfig.setValue({
11             fieldId: 'REVERSALVOIDING',
12             value: false
13         });
14
15         accountingConfig.save();
16
17         var salesOrderObj = record.create({
18             type: 'salesorder',
19             isDynamic: false
20         });
21         salesOrderObj.setValue({
22             fieldId: 'entity',
23             value: 107
24         });
25         salesOrderObj.setSublistValue({
26             sublistId: 'item',
27             fieldId: 'item',
28             value: 233,
29             line: 0
30         });
31         salesOrderObj.setSublistValue({
32             sublistId: 'item',
33             fieldId: 'amount',
34             value: 1,
35             line: 0
36         });
37
38         var salesOrderId = salesOrderObj.save();
39
40         var voidSalesOrderId = transaction.void({
41             type: record.Type.SALES_ORDER,
42             id: salesOrderId
43         });
44
45         var salesOrder = record.load({
46             type: 'salesorder',
47             id: voidSalesOrderId
48         });
49
50         // The value of the memo field is 'VOID'
51         var memo = salesOrder.getValue({

```

```

52         fieldId: 'memo'
53     });
54 }
55
56 voidSalesOrder();
57 });

```

N/translation Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/translation Module](#):

- [Access Parameterized Translation Strings](#)
- [Access Translation Strings](#)
- [Access Translation Strings Using a Non-Default Locale](#)
- [Load Specific Translation Strings from a Collection](#)
- [Load Translation Strings by Key from Multiple Translation Collections](#)
- [Load Translation Strings by Key from a Translation Collection with Multiple Locales](#)

Access Parameterized Translation Strings

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample accesses parametrized translation strings. When you create a Translation Collection in the NetSuite UI, you can include parameter placeholders in your translation strings. Placeholders use braces and a number (starting from 1). The translator function injects the specified parameter values into the placeholders in the translation string. For example, “Hello, {1}!” is a valid translation string, where {1} is a placeholder for a parameter. In this sample, the parameter “NetSuite” is provided to the translator function returned from [translation.get\(options\)](#), and the translator function returns a translated string of “Hello, NetSuite!”

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/ui/message', 'N/translation'],
6    function(message, translation) {
7
8      // Create a message with translated strings
9      var myMsg = message.create({
10         title: translation.get({
11             collection: 'custcollection_my_strings',
12             key: 'MY_TITLE'
13         })(),
14         message: translation.get({
15             collection: 'custcollection_my_strings',

```

```

16         key: 'HELLO_1'
17     )}{
18         params: ['NetSuite']
19     },
20     type: message.Type.CONFIRMATION
21 });
22
23 // Show the message for 5 seconds
24 myMsg.show({
25     duration: 5000
26 });
27 });

```

Access Translation Strings

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample accesses translation strings one at a time using [translation.get\(options\)](#). This method returns a translator function, which is subsequently called with any specified parameters. The translator function returns the string in the user's session locale by default.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/ui/message', 'N/translation'],
6      function(message, translation) {
7
8          // Create a message with translated strings
9          var myMsg = message.create({
10             title: translation.get({
11                 collection: 'custcollection_my_strings',
12                 key: 'MY_TITLE'
13             })(),
14             message: translation.get({
15                 collection: 'custcollection_my_strings',
16                 key: 'MY_MESSAGE'
17             })(),
18             type: message.Type.CONFIRMATION
19         });
20
21         // Show the message for 5 seconds
22         myMsg.show({
23             duration: 5000
24         });
25     });

```

Access Translation Strings Using a Non-Default Locale

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample accesses translation strings using a locale other than the default locale. When you call [translation.get\(options\)](#) and do not specify a locale, the method uses the current user's session locale.

You can use the `options.locale` parameter to specify another locale. The [translation.Locale](#) enum lists all locales that are enabled for a company, and you can use these locales in `translation.get(options)`. The `translation.Locale` enum also includes two special values: `CURRENT` and `COMPANY_DEFAULT`. The `CURRENT` value represents the current user's locale, and the `COMPANY_DEFAULT` value represents the default locale for the company.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/ui/message', 'N/translation'],
6    function(message, translation) {
7
8      // Create a message with translated strings
9      var myMsg = message.create({
10         title: translation.get({
11             collection: 'custcollection_my_strings',
12             key: 'MY_TITLE',
13             locale: translation.Locale.COMPANY_DEFAULT
14         })(),
15         message: translation.get({
16             collection: 'custcollection_my_strings',
17             key: 'MY_MESSAGE',
18             locale: translation.Locale.COMPANY_DEFAULT
19         })(),
20         type: message.Type.CONFIRMATION
21     });
22
23     // Show the message for 5 seconds
24     myMsg.show({
25         duration: 5000
26     });
27 });

```

Load Specific Translation Strings from a Collection

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads specific translation strings from a collection. The [translation.load\(options\)](#) method can load a maximum of 1,000 translation strings. If you need only a few of the translation strings in a collection, you can load only the strings you need instead of loading the entire collection.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/ui/message', 'N/translation'],
6    function(message, translation) {

```



```

7
8      // Load translation strings by key
9      var localizedStrings = translation.load({
10         collections: [{
11             alias: 'myCollection',
12             collection: 'custcollection_my_strings',
13             keys: ['MY_TITLE', 'MY_MESSAGE']
14         }]
15     });
16
17     // Create a message with translated strings
18     var myMsg = message.create({
19         title: localizedStrings.myCollection.MY_TITLE(),
20         message: localizedStrings.myCollection.MY_MESSAGE(),
21         type: message.Type.CONFIRMATION
22     });
23
24     // Show the message for 5 seconds
25     myMsg.show({
26         duration: 5000
27     });
28 });

```

Load Translation Strings by Key from Multiple Translation Collections

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads translation strings by key from multiple Translation Collections in a single call of [translation.load\(options\)](#). This method can load a maximum of 1,000 translation strings, regardless of whether the strings are loaded from one collection or multiple collections.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5  require(['N/ui/message', 'N/translation'],
6      function(message, translation) {
7
8      // Load two Translation Collections
9      var localizedStrings = translation.load({
10         collections: [{
11             alias: 'myCollection',
12             collection: 'custcollection_my_strings',
13             keys: ['MY_TITLE']
14         }, {
15             alias: 'myOtherCollection',
16             collection: 'custcollection_other_strings',
17             keys: ['MY_OTHER_MESSAGE']
18         }]
19     });
20
21     // Create a message with translated strings
22     var myMsg = message.create({
23         title: localizedStrings.myCollection.MY_TITLE(),
24         message: localizedStrings.myOtherCollection.MY_OTHER_MESSAGE(),
25         type: message.Type.CONFIRMATION

```

```

26     });
27
28     // Show the message for 5 seconds
29     myMsg.show({
30         duration: 5000
31     });
32 });

```

Load Translation Strings by Key from a Translation Collection with Multiple Locales

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads translation strings by key from a Translation Collection with multiple locales. When you load translation strings using [translation.load\(options\)](#), you can specify a list of valid locales for the strings. You can use these locales when you select a locale using [translation.selectLocale\(options\)](#). If you specify more than one locale when you call `translation.load(options)`, the first specified locale in the list is used for the created [translation.Handle](#) object. If you want to use a different locale from the list, use `translation.selectLocale(options)`, which returns a `translation.Handle` object in the specified locale. You must load a locale using `translation.load(options)` before you can select it using `translation.selectLocale(options)`.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   require(['N/ui/message', 'N/translation'],
6     function(message, translation) {
7
8       // Load a Translation Collection and a set of locales
9       var germanStrings = translation.load({
10         collections: [{
11             alias: 'myCollection',
12             collection: 'custcollection_my_strings',
13             keys: ['MY_TITLE', 'MY_MESSAGE']
14         }],
15         locales: [translation.Locale.de_DE, translation.Locale.es_ES]
16     });
17
18     // Select a locale from the list of loaded locales
19     var spanishStrings = translation.selectLocale({
20         handle: germanStrings,
21         locale: translation.Locale.es_ES
22     });
23
24     // Create a message with translated strings
25     var myMsg = message.create({
26         title: germanStrings.myCollection.MY_TITLE(),
27         message: spanishStrings.myCollection.MY_MESSAGE(),
28         type: message.Type.CONFIRMATION
29     });
30
31     // Show the message for 5 seconds
32     myMsg.show({
33         duration: 5000
34     });

```

```
35 | });
```

N/ui/dialog Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/ui/dialog Module](#):

- [Create a Confirmation Dialog](#)
- [Create a Dialog with Buttons](#)
- [Create a Dialog that Includes a Default Button](#)
- [Create an Alert Dialog](#)

Create a Confirmation Dialog

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a confirmation dialog.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

i Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  define(['N/ui/dialog'], (dialog) => {
5      let options = {
6          title: 'I am a Confirmation',
7          message: 'Press OK or Cancel'
8      };
9
10     function success(result) {
11         console.log('Success with value ' + result);
12     }
13
14     function failure(reason) {
15         console.log('Failure: ' + reason);
16     }
17
18     dialog.confirm(options).then(success).catch(failure);
19 });

```

Create a Dialog with Buttons

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a dialog with three buttons.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4   define(['N/ui/dialog'], (dialog) => {
5       let button1 = {
6           label: 'I am A',
7           value: 1
8       };
9       let button2 = {
10          label: 'I am B',
11          value: 2
12      };
13      let button3 = {
14          label: 'I am C',
15          value: 3
16      };
17      let options = {
18          title: 'Alphabet Test',
19          message: 'Which One?',
20          buttons: [button1, button2, button3]
21      };
22
23      function success(result) {
24          console.log('Success with value ' + result);
25      }
26      function failure(reason) {
27          console.log('Failure: ' + reason);
28      }
29      dialog.create(options).then(success).catch(failure);
30  });

```

Create a Dialog that Includes a Default Button

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a dialog without specifying any buttons. The default behavior is to create a dialog that includes a single button with the label OK.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  define(['N/ui/dialog'], (dialog) => {
5      let options = {
6          title: 'I am a Dialog with the default button',
7          message: 'Click a button to continue.',
8      };
9
10     function success(result) {
11         console.log('Success with value ' + result);
12     }
13
14     function failure(reason) {
15         console.log('Failure: ' + reason);
16     }
17
18     dialog.create(options).then(success).catch(failure);
19 });

```

Create an Alert Dialog

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create an alert dialog.

Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  define(['N/ui/dialog'], (dialog) => {
5      let options = {
6          title: 'I am an Alert',

```

```

7      message: 'Click OK to continue.'
8    };
9
10   function success(result) {
11     console.log('Success with value ' + result);
12   }
13
14   function failure(reason) {
15     console.log('Failure: ' + reason);
16   }
17
18   dialog.alert(options).then(success).catch(failure);
19 });

```

N/ui/message Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/ui/message Module](#).

■ Create Confirmation, Information, Warning, and Error Messages

Create Confirmation, Information, Warning, and Error Messages

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create messages for the four available types (confirmation, information, warning, and error).

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

i Note: To debug client scripts like the following, you should use Chrome DevTools for Chrome, Firebug debugger for Firefox, or Microsoft Script Debugger for Internet Explorer. For information about these tools, see the documentation provided with each browser. For more information about debugging SuiteScript client scripts, see the help topic [Debugging Client Scripts](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4  define(['N/ui/message'], (message) => {
5    let myMsg = message.create({
6      title: 'My Title',
7      message: 'My Message',
8      type: message.Type.CONFIRMATION
9    });
10   myMsg.show({
11     duration: 5000 // will disappear after 5s
12   });
13
14   let myMsg2 = message.create({
15     title: 'My Title 2',
16     message: 'My Message 2',

```

```

17     type: message.Type.INFORMATION
18   });
19   myMsg2.show();
20   setTimeout(myMsg2.hide, 15000); // will disappear after 15s
21
22   let myMsg3 = message.create({
23     title: 'My Title 3',
24     message: 'My Message 3',
25     type: message.Type.WARNING,
26     duration: 20000
27   });
28   myMsg3.show(); // will disappear after 20s
29
30   let myMsg4 = message.create({
31     title: 'My Title 4',
32     message: 'My Message 4',
33     type: message.Type.ERROR
34   });
35   myMsg4.show(); // will stay up until hide is called.
36 });

```

N/ui/serverWidget Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/ui/serverWidget Module](#):

- [Create a Custom Form with a Submit Button, Fields, and an Inline Editor Sublist](#)
- [Create a Custom Survey Form](#)

Create a Custom Form with a Submit Button, Fields, and an Inline Editor Sublist

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a Suitelet that generates a sample form with a submit button, fields, and an inline editor sublist.

i Note: This script sample uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

⚠ Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType Suitelet
4   */
5   define(['N/ui/serverWidget'], (serverWidget) => {
6     const onRequest = (scriptContext) => {
7       if (scriptContext.request.method === 'GET') {
8         let form = serverWidget.createForm({
9           title: 'Simple Form'
10        });
11
12        let field = form.addField({
13          id: 'textfield',
14          type: serverWidget.FieldType.TEXT,
15          label: 'Text'

```

```

16     });
17     field.layoutType = serverWidget.FieldLayoutType.NORMAL;
18     field.updateBreakType({
19         breakType: serverWidget.FieldBreakType.STARTCOL
20     });
21
22     form.addField({
23         id: 'datefield',
24         type: serverWidget.FieldType.DATE,
25         label: 'Date'
26     });
27     form.addField({
28         id: 'currencyfield',
29         type: serverWidget.FieldType.CURRENCY,
30         label: 'Currency'
31     });
32
33     let select = form.addField({
34         id: 'selectfield',
35         type: serverWidget.FieldType.SELECT,
36         label: 'Select'
37     });
38     select.addSelectOption({
39         value: 'a',
40         text: 'Albert'
41     });
42     select.addSelectOption({
43         value: 'b',
44         text: 'Baron'
45     });
46
47     let sublist = form.addSublist({
48         id: 'sublist',
49         type: serverWidget.SublistType.INLINEEDITOR,
50         label: 'Inline Editor Sublist'
51     });
52     sublist.addField({
53         id: 'sublist1',
54         type: serverWidget.FieldType.DATE,
55         label: 'Date'
56     });
57     sublist.addField({
58         id: 'sublist2',
59         type: serverWidget.FieldType.TEXT,
60         label: 'Text'
61     });
62
63     form.addSubmitButton({
64         label: 'Submit Button'
65     });
66
67     scriptContext.response.writePage(form);
68 } else {
69     const delimiter = /\u0001/;
70     const textField = scriptContext.request.parameters.textfield;
71     const dateField = scriptContext.request.parameters.datefield;
72     const currencyField = scriptContext.request.parameters.currencyfield;
73     const selectField = scriptContext.request.parameters.selectfield;
74     const sublistData = scriptContext.request.parameters.sublistdata.split(delimiter);
75     const sublistField1 = sublistData[0];
76     const sublistField2 = sublistData[1];
77
78     scriptContext.response.write('You have entered: ${textField} ${dateField} ${currencyField} ${selectField} ${sublist
Field1} ${sublistField2}');
79 }
80 }
81
82 return {onRequest}
83 });

```


Create a Custom Survey Form

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a Suitelet that generates a customer survey form with inline HTML fields, radio fields, and a submit button.

i Note: This script sample uses the define function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the require function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript 2.x Global Objects](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5  define(['N/ui/serverWidget'], function(serverWidget) {
6      function onRequest(context) {
7          var form = serverWidget.createForm({
8              title: 'Thank you for your interest in Wolfe Electronics',
9              hideNavBar: true
10         });
11
12         var htmlHeader = form.addField({
13             id: 'custpage_header',
14             type: serverWidget.FieldType.INLINEHTML,
15             label: ' '
16         }).updateLayoutType({
17             layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
18         }).updateBreakType({
19             breakType: serverWidget.FieldBreakType.STARTROW
20         }).defaultValue = '<p style=\'font-size:20px\'>We pride ourselves on providing the best' +
21             ' services and customer satisfaction. Please take a moment to fill out our survey.</p><br><br>';
22
23         var htmlInstruct = form.addField({
24             id: 'custpage_p1',
25             type: serverWidget.FieldType.INLINEHTML,
26             label: ' '
27         }).updateLayoutType({
28             layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
29         }).updateBreakType({
30             breakType: serverWidget.FieldBreakType.STARTROW
31         }).defaultValue = '<p style=\'font-size:14px\'>When answering questions on a scale of 1 to 10, ' +
32             ' 1 = Greatly Unsatisfied and 10 = Greatly Satisfied.</p><br><br><br>';
33
34         var productRating = form.addField({
35             id: 'custpage_lblproductrating',
36             type: serverWidget.FieldType.INLINEHTML,
37             label: ' '
38         }).updateLayoutType({
39             layoutType: serverWidget.FieldLayoutType.NORMAL
40         }).updateBreakType({
41             breakType: serverWidget.FieldBreakType.STARTROW
42         }).defaultValue = '<p style=\'font-size:14px\'>How would you rate your satisfaction with our products?</p>';
43
44         form.addField({
45             id: 'custpage_idoproducrating',
46             type: serverWidget.FieldType.RADIO,
47             label: '1',
48             source: 'p1'
49         }).updateLayoutType({
50             layoutType: serverWidget.FieldLayoutType.STARTROW
51         });
52         form.addField({
53             id: 'custpage_idoproducrating',
54             type: serverWidget.FieldType.RADIO,
55             label: '2',
56             source: 'p2'
57         }).updateLayoutType({

```

```

58         layoutType: serverWidget.FieldLayoutType.MIDROW
59     });
60     form.addField({
61         id: 'custpage_rdoproducrating',
62         type: serverWidget.FieldType.RADIO,
63         label: '3',
64         source: 'p3'
65     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
66     form.addField({
67         id: 'custpage_rdoproducrating',
68         type: serverWidget.FieldType.RADIO,
69         label: '4',
70         source: 'p4'
71     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
72     form.addField({
73         id: 'custpage_rdoproducrating',
74         type: serverWidget.FieldType.RADIO,
75         label: '5',
76         source: 'p5'
77     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
78     form.addField({
79         id: 'custpage_rdoproducrating',
80         type: serverWidget.FieldType.RADIO,
81         label: '6',
82         source: 'p6'
83     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
84     form.addField({
85         id: 'custpage_rdoproducrating',
86         type: serverWidget.FieldType.RADIO,
87         label: '7',
88         source: 'p7'
89     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
90     form.addField({
91         id: 'custpage_rdoproducrating',
92         type: serverWidget.FieldType.RADIO,
93         label: '8',
94         source: 'p8'
95     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
96     form.addField({
97         id: 'custpage_rdoproducrating',
98         type: serverWidget.FieldType.RADIO,
99         label: '9',
100        source: 'p9'
101    }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
102    form.addField({
103        id: 'custpage_rdoproducrating',
104        type: serverWidget.FieldType.RADIO,
105        label: '10',
106        source: 'p10'
107    }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.ENDROW});
108
109    var serviceRating = form.addField({
110        id: 'custpage_lblservicerating',
111        type: serverWidget.FieldType.INLINEHTML,
112        label: ' '
113    }).updateLayoutType({
114        layoutType: serverWidget.FieldLayoutType.NORMAL
115    }).updateBreakType({
116        breakType: serverWidget.FieldBreakType.STARTROW
117    }).defaultValue = '<p style=\"font-size:14px\">How would you rate your satisfaction with our services?</p>';
118
119    form.addField({
120        id: 'custpage_rdoservicerating',
121        type: serverWidget.FieldType.RADIO,
122        label: '1',
123        source: 'p1'
124    }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.STARTROW});
125    form.addField({
126        id: 'custpage_rdoservicerating',
127        type: serverWidget.FieldType.RADIO,
128        label: '2',
129        source: 'p2'
130    }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});

```

```

131     form.addField({
132         id: 'custpage_rdoservicerating',
133         type: serverWidget.FieldType.RADIO,
134         label: '3',
135         source: 'p3'
136     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
137     form.addField({
138         id: 'custpage_rdoservicerating',
139         type: serverWidget.FieldType.RADIO,
140         label: '4',
141         source: 'p4'
142     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
143     form.addField({
144         id: 'custpage_rdoservicerating',
145         type: serverWidget.FieldType.RADIO,
146         label: '5',
147         source: 'p5'
148     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
149     form.addField({
150         id: 'custpage_rdoservicerating',
151         type: serverWidget.FieldType.RADIO,
152         label: '6',
153         source: 'p6'
154     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
155     form.addField({
156         id: 'custpage_rdoservicerating',
157         type: serverWidget.FieldType.RADIO,
158         label: '7',
159         source: 'p7'
160     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
161     form.addField({
162         id: 'custpage_rdoservicerating',
163         type: serverWidget.FieldType.RADIO,
164         label: '8',
165         source: 'p8'
166     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
167     form.addField({
168         id: 'custpage_rdoservicerating',
169         type: serverWidget.FieldType.RADIO,
170         label: '9',
171         source: 'p9'
172     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
173     form.addField({
174         id: 'custpage_rdoservicerating',
175         type: serverWidget.FieldType.RADIO,
176         label: '10',
177         source: 'p10'
178     }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.ENDROW});
179
180     form.addSubmitButton({
181         label: 'Submit'
182     });
183
184     context.response.writePage(form);
185 }
186
187 return {
188     onRequest: onRequest
189 };
190 });

```

N/url Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/url Module](#):

- [Create a URL and Send a Secure HTTPS Post Request to the URL](#)

- [Generate an Absolute URL to a Specific Resource](#)
- [Retrieve the Domain for Calling a RESTlet](#)
- [Retrieve the Relative URL of a Record](#)

Create a URL and Send a Secure HTTPS Post Request to the URL

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a URL and send a secure HTTPS POST request to that URL with an empty body. The server's response is also logged.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: The value used in this sample for the `scriptId` and `deploymentId` fields are placeholders. Before using this sample, replace the `scriptId` and `deploymentId` values with valid values from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script creates a URL, sends a secure HTTPS POST request to that URL, and logs the server's response.
6   require(['N/url', 'N/https'], function(url, https) {
7       var script = 'customscript1';
8       var deployment = 'customdeploy1';
9       var parameters = '';
10      try {
11          var suiteletURL = url.resolveScript({
12              scriptId: script,
13              deploymentId: deployment
14          });
15          var response = https.post({
16              url: suiteletURL,
17              body: parameters
18          });
19          log.debug(response.body.toString());
20      }
21      catch(e) {
22          log.error(e.toString());
23      }
24  });

```

Generate an Absolute URL to a Specific Resource

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to generate an absolute URL to a specific resource.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: The value used in this sample for the `recordId` field is a placeholder. Before using this sample, replace the `recordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script generates an absolute URL to a specific resource.
6   require(['N/url', 'N/record'], function(url, record) {
7       function resolveRecordUrl() {
8           var scheme = 'https://';
9           var host = url.resolveDomain({
10              hostType: url.HostType.APPLICATION
11            });
12           var relativePath = url.resolveRecord({
13              recordType: record.Type.SALES_ORDER,
14              recordId: 6,
15              isEditMode: true
16            });
17           var myURL = scheme + host + relativePath;
18       }
19       resolveRecordUrl();
20   });

```

Retrieve the Domain for Calling a RESTlet

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to retrieve the domain for calling a RESTlet.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: The value used in this sample for the `accountId` field is a placeholder. Before using this sample, replace the `accountId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script retrieves the domain for calling a RESTlet.
6   require(['N/url'], function(url) {
7       function resolveDomainUrl() {
8           var sCompId = 'MSTRWLF';
9           var output = url.resolveDomain({
10              hostType: url.HostType.RESTLET,
11              accountId: sCompId
12            });
13       }
14       resolveDomainUrl();
15   });

```

Retrieve the Relative URL of a Record

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to retrieve the relative URL of a record. With the internal ID value used in this sample, the returned output is `/app/accounting/transactions/salesord.nl?id=6&e=T&compid=`, followed by the NetSuite account ID.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: The value used in this sample for the `recordId` field is a placeholder. Before using this sample, replace the `recordId` field value with a valid value from your NetSuite account. If you run a script with an invalid value, an error may occur.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   // This script retrieves the relative URL of a record.
6   require(['N/url'], function(url) {
7       var output = url.resolveRecord({
8           recordType: 'salesorder',
9           recordId: 6,
10          isEditMode: true
11      });
12  });

```

N/util Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/util Module](#):

- [Set Fields on a Sales Order Record Using the util.each Iterator](#)

Set Fields on a Sales Order Record Using the util.each Iterator

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a sales order record. It uses the `util.each(iterable, callback)` method to set record fields based on the values in an iterable object. Be sure to replace hard-coded values (such as record IDs) with valid values from your NetSuite account.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */

```

```

4
5 require(['N/record'], function(record){
6     // Create a sales order
7     var rec = record.create({
8         type: 'salesorder',
9         isDynamic: true
10    });
11    rec.setValue({
12        fieldId: 'entity',
13        value: 107
14    });
15
16    // Set up an object containing an item's internal ID and the corresponding quantity
17    var itemList = {
18        39: 5,
19        38: 1
20    }
21
22    // Iterate through the object and set the key-value pairs on the record
23    util.each(itemList, function(quantity, itemId){ // (5, 39) and (1, 38)
24        rec.selectNewLine('item');
25        rec.setCurrentSublistValue('item', 'item', itemId);
26        rec.setCurrentSublistValue('item', 'quantity', quantity);
27        rec.commitLine('item');
28    });
29
30    var id = rec.save();
31 });

```

N/workbook Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/workbook Module](#):

- [Create Datasets, Dataset Links, and a Workbook with a Pivot and Run the Workbook](#)
- [Create a Comprehensive Workbook](#)

Create Datasets, Dataset Links, and a Workbook with a Pivot and Run the Workbook

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates two datasets, links them, and creates a workbook with a pivot based on the data in the datasets.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */
4   // The following script creates a simple workbook that contains a simple table
5   require(['N/workbook', 'N/dataset', 'N/datasetLink'], function(nWorkbook, nDataset, datasetLink) {
6       var join = nDataset.createJoin({
7           fieldId: "budgetmachine"
8       });
9
10      var period = nDataset.createColumn({
11          join: join,

```

```

12     fieldId: "period",
13     alias: "budgetmachineperiod",
14     label: "Accounting Period"
15 });
16
17 var department = nDataset.createColumn({
18     fieldId: "department",
19     alias: "department",
20     label: "Department"
21 });
22
23 var total = nDataset.createColumn({
24     fieldId: "total",
25     alias: "total",
26     label: "Amount (Total)"
27 });
28
29 var budget = nDataset.create({
30     type: 'budgets',
31     columns: [period, department, total]
32 });
33
34 var postingperiod = nDataset.createColumn({
35     fieldId: "postingperiod",
36     alias: "postingperiod",
37     label: "Posting Period"
38 });
39
40 var amount = nDataset.createColumn({
41     fieldId: "amount",
42     alias: "amount",
43     label: "Amount"
44 });
45
46 var sales = nDataset.create({
47     type: 'salesinvoiced',
48     columns: [postingperiod, department, amount],
49 });
50
51
52 var budgetmachineperiod = budget.getExpressionFromColumn({
53     alias: "budgetmachineperiod"
54 });
55 var postingperiodExpression = sales.getExpressionFromColumn({
56     alias: "postingperiod"
57 });
58 var link = datasetLink.create({
59     datasets: [budget, sales],
60     expressions: [[budgetmachineperiod, postingperiodExpression]],
61     id: "link"
62 });
63
64 var postingPeriodItem = nWorkbook.createDataDimensionItem({
65     expression: postingperiodExpression
66 });
67 var postingPeriodDimension = nWorkbook.createDataDimension({
68     items: [postingPeriodItem]
69 });
70 var rowSection = nWorkbook.createSection({
71     children: [postingPeriodDimension]
72 });
73
74 var departmentItem = nWorkbook.createDataDimensionItem({
75     expression: budget.getExpressionFromColumn({
76         alias: "department"
77     })
78 });
79 var departmentDimension = nWorkbook.createDataDimension({
80     items: [departmentItem]
81 });
82
83 var sumTotal = nWorkbook.createDataMeasure({
84     label: 'Sum Total',

```



```

85     expression: budget.getExpressionFromColumn({
86         alias: 'total'
87     }),
88     aggregation: 'SUM'
89 });
90
91 var sumAmountNet = nWorkbook.createDataMeasure({
92     label: 'Sum Amount',
93     expression: sales.getExpressionFromColumn({
94         alias: 'amount'
95     }),
96     aggregation: 'SUM'
97 });
98
99 var columnSection = nWorkbook.createSection({
100     children: [departmentDimension, sumTotal, sumAmountNet]
101 });
102
103 var pivot = nWorkbook.createPivot({
104     id: "pivot",
105     rowAxis: nWorkbook.createPivotAxis({
106         root: rowSection
107     }),
108     columnAxis: nWorkbook.createPivotAxis({
109         root: columnSection
110     }),
111     name: "Pivot",
112     datasetLink: link
113 });
114
115 var wb = nWorkbook.create({
116     pivots: [pivot]
117 });
118
119 wb.runPivot.promise("pivot").then(function(intersections){
120     for (var i in intersections)
121     {
122         var intersection = intersections[i];
123         if (intersection.row.itemValues) //skip header
124         {
125             console.log("Period: " + intersection.row.itemValues[0].value.name);
126             console.log(intersection.column.section.children[1].label + ":");
127             console.log(intersection.measureValues[0] ? intersection.measureValues[0].value.amount : 0);
128             console.log(intersection.column.section.children[2].label + ":");
129             console.log(intersection.measureValues[1] ? intersection.measureValues[1].value.amount : 0);
130         }
131     }
132 })
133 });

```

Create a Comprehensive Workbook

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample creates a workbook that includes a chart, a table, and a pivot. This sample uses a dataset that is not included in your account, so you will need to change the dataset id to a valid value from your account. This dataset needs to include columns for 'id', 'name', 'date', and 'total'.

i Note: This sample script uses the `require` function so you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (a script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   */

```

```

4 // The following script creates a workbook that includes a chart, a table, and a pivot. This sample uses a dataset that is not included in your account, so
  // you will need to change the dataset id to a valid value from your account. This dataset needs to include columns for 'id', 'name', 'date', and 'total'.
5
6 require(['N/workbook', 'N/dataset'], function(workbook, dataset){
7     var myDataset = dataset.load({
8         id: 'dataset_7'
9     });
10
11     var theIDExpression = myDataset.getExpressionFromColumn({
12         alias: 'id'
13     });
14     var sort = workbook.createSort({
15         ascending: false
16     });
17     var columnID = workbook.createTableColumn({
18         datasetColumnAlias: 'id'
19     });
20     var columnName = workbook.createTableColumn({
21         datasetColumnAlias: 'name'
22     });
23     var columnDate = workbook.createTableColumn({
24         datasetColumnAlias: 'date'
25     });
26     var columnTotal = myBasicWorkbook.createTableColumn({
27         datasetColumnAlias: 'total'
28     });
29     var tableView = workbook.createTableDefinition({
30         id: 'view',
31         name: 'View',
32         dataset: myDataset,
33         columns: [columnID, columnName, columnDate, columnTotal]
34     });
35     var theDateExpression = dataset.getExpressionFromColumn({
36         alias: 'date'
37     });
38     var rowItem = workbook.createDataDimensionItem({
39         label: 'A',
40         expression: theDateExpression
41     });
42     var rowDataDimension = workbook.createDataDimension({
43         items: [rowItem]
44     });
45     var rowSection = workbook.createSection({
46         children: [rowDataDimension]
47     });
48     var theTotalExpression = dataset.getExpressionFromColumn({
49         alias: 'total'
50     });
51     var columnItem = workbook.createDataDimensionItem({
52         label: 'B',
53         expression: theTotalExpression
54     }); var columnDataDimension = workbook.createDataDimension({
55         items: [columnItem]
56     });
57     var columnMeasure = workbook.createMeasure({
58         label: 'M',
59         expression: theIDExpression,
60         aggregation: workbook.Aggregation.MAX
61     });
62     var columnSection = workbook.createSection({
63         children: [columnDataDimension, columnMeasure]
64     });
65     var constExpr = workbook.createConstant({
66         constant: 1
67     });
68     var anyOfExpr = workbook.createExpression({
69         functionId: workbook.ExpressionType.AND,
70         parameters: {
71             expression: theIDExpression,
72             set: [constExpr]
73         }
74     });
75     var notExpr = workbook.createExpression({

```

```

76         functionId: workbook.ExpressionType.NOT,
77         parameters: {
78             a: anyOfExpr
79         }
80     });
81     var allSubNodesSelector = workbook.createAllSubNodesSelector();
82     var rowItemSelector = workbook.createDimensionSelector({
83         dimension: rowDataDimension
84     });
85     var columnItemSelector = workbook.createDimensionSelector({
86         dimension: columnDataDimension
87     });
88     var rowSelector = workbook.createPathSelector({
89         elements: [allSubNodesSelector, rowItemSelector]
90     });
91     var columnSelector = workbook.createPathSelector({
92         elements: [allSubNodesSelector, columnItemSelector]
93     });
94     var rowSort = workbook.createDimensionSort({
95         item: rowItem,
96         sort: sort
97     });
98     var columnSort = workbook.createMeasureSort({
99         measure: columnMeasure,
100         sort: sort,
101         otherAxisSelector: allSubNodesSelector
102     });
103     var rowSortDefinition = workbook.createSortDefinition({
104         sortBy: [rowSort],
105         selector: rowSelector
106     });
107     var columnSortDefinition = workbook.createSortDefinition({
108         sortBy: [columnSort],
109         selector: columnSelector
110     });
111     var rowAxis = workbook.createPivotAxis({
112         root: rowSection,
113         sortDefinitions: [rowSortDefinition]
114     });
115     var columnAxis = workbook.createPivotAxis({
116         root: columnSection,
117         sortDefinitions: [columnSortDefinition]
118     });
119     var limitingFilter = workbook.createLimitingFilter({
120         row: true,
121         filteredNodesSelector: rowSelector,
122         limit: 1,
123         sortBy: [rowSort]
124     });
125     var conditionalFilter = workbook.createConditionalFilter({
126         row: false,
127         filteredNodesSelector: rowSelector,
128         otherAxisSelector: columnSelector,
129         measure: columnMeasure,
130         predicate: notExpr
131     });
132     var pivot = workbook.createPivotDefinition({
133         id: 'pivot',
134         name: 'Pivot',
135         dataset: myDataset,
136         rowAxis: rowAxis,
137         columnAxis: columnAxis,
138         filterExpressions: [notExpr],
139         aggregationFilters: [limitingFilter, conditionalFilter]
140     });
141     var firstAxis = workbook.createChartAxis({
142         title: 'First axis'
143     });
144     var secondAxis = workbook.createChartAxis({
145         title: 'Second axis'
146     });
147     var category = workbook.createCategory({
148         axis: firstAxis,

```

```

149     root: rowSection
150   });
151   var legend = workbook.createLegend({
152     axes: [secondAxis],
153     root: columnSection
154   });
155   var aspect = workbook.createAspect({
156     measure: columnMeasure
157   });
158   var series = workbook.createSeries({
159     aspects: [aspect]
160   });
161   var chart = workbook.createChartDefinition({
162     id: 'chart',
163     name: 'Chart',
164     type: workbook.ChartType.AREA,
165     dataset: myDataset,
166     category: category,
167     legend: legend,
168     series: [series]
169   });
170   var myNewWorkbook = workbook.create({
171     description: 'My new updated workbook',
172     name: 'Workbook Updated',
173     tableDefinitions: [tableView],
174     pivotDefinitions: [pivot],
175     chartDefinitions: [chart]
176   });
177   var workbookList = workbook.list();
178
179   log.debug({
180     title: "MyNewWorkbook",
181     details: myNewWorkbook
182   });
183 });

```

N/workflow Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code sample is provided for the [N/workflow Module](#):

- [Search For and Execute a Workflow Deployed on the Customer Record](#)

Search For and Execute a Workflow Deployed on the Customer Record

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample searches for a specific workflow deployed on the customer record and then executes it.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

Important: This script sample uses placeholder values for the customer `recordId` and `workflowId`. Before using this sample, replace these IDs with valid values from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

```

1  /**
2   * @NApiVersion 2.x
3   */
4
5   require(['N/workflow', 'N/search', 'N/error', 'N/record'],
6     function(workflow, search, error, record) {
7       function initiateWorkflow() {
8         var workflowInstanceId = workflow.initiate({
9           recordType: 'customer',
10          recordId: 24,
11          workflowId: 'customworkflow_myWorkflow'
12        });
13        var customerRecord = record.load({
14          type: record.Type.CUSTOMER,
15          id: 24
16        });
17      }
18      initiateWorkflow();
19    });

```

N/xml Samples

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following code samples are provided for the [N/xml Module](#):

- [Load an XML File and Obtain Child Element Values](#)
- [Load an XML File from the File Cabinet and Sign It using a Digital Certificate](#)
- [Modify an XML File](#)
- [Parse an XML String and Log Element Values](#)
- [Render an Invoice Into a PDF Using an XML Template](#)

Load an XML File and Obtain Child Element Values

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads the `BookSample.xml` file from the File Cabinet, iterates through the individual book nodes, and accesses the child node values.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**

```

```

2  * @NApiVersion 2.x
3  * @NScriptType Suitelet
4  */
5  require(['N/xml', 'N/file'], function(xml, file) {
6      return {
7          onRequest: function(options) {
8              var sentence = '';
9              var xmlFileContent = file.load('SuiteScripts/BookSample.xml').getContents();
10             var xmlDocument = xml.Parser.fromString({
11                 text: xmlFileContent
12             });
13             var bookNode = xml.XPath.select({
14                 node: xmlDocument,
15                 xpath: '//b:book'
16             });
17
18             for (var i = 0; i < bookNode.length; i++) {
19                 var title = bookNode[i].firstChild.nextSibling.textContent;
20                 var author = bookNode[i].getElementsByName({
21                     tagName: 'b:author'
22                 })[0].textContent;
23                 sentence += 'Author: ' + author + ' wrote ' + title + '.\n';
24             }
25
26             options.response.write(sentence);
27         }
28     };
29 });

```

This script produces the following output when used with the BookSample.xml file:

```

1  Author: Giada De Laurentiis wrote Everyday Italian.
2  Author: J K. Rowling wrote Harry Potter.
3  Author: James McGovern wrote XQuery Kick Start.
4  Author: Erik T. Ray wrote Learning XML.

```

Load an XML File from the File Cabinet and Sign It using a Digital Certificate

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample loads an XML file from the File Cabinet and signs it using the digital certificate with internal ID 'custcertificate1'. Note that this sample uses a hard-coded value for the file id. You should change this value to a valid file id from your account.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

! Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   */
4
5  require(['N/crypto/certificate', 'N/file'], (cert, file) => {

```

```

6 // Load the file from the File Cabinet.
7 // Note that the id value is hard-coded in this sample, and you should use
8 // a valid file id from your account.
9 let infNFe = file.load({
10   id: 922
11 });
12 let signedXml = cert.signXml({
13   algorithm: certificate.HashAlg.SHA256,
14   certId: 'custcertificate1',
15   rootTag: 'infNFe',
16   xmlString: infNFe.getContents()
17 });
18 cert.verifyXMLSignature({
19   signedXml: signedXml,
20   rootTag: 'infNFe'
21 });
22 }); // Note that this value is hard-coded in this sample, and you should use

```

Modify an XML File

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to modify an XML file.

i Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1 /**
2  * @NApiVersion 2.x
3  */
4
5 require(['N/xml', 'N/file'], function(xml, file) {
6   var xmlData = file.load('SuiteScripts/BookSample.xml').getContents();
7   var bookShelf = xml.Parser.fromString({
8     text: xmlData
9   });
10
11   var newBookNode = bookShelf.createElement("book");
12   var newTitleNode = bookShelf.createElement("title");
13   var newTitleNodeValue = bookShelf.createTextNode("");
14   var newAuthorNode = bookShelf.createElement("author");
15   var newAuthorNodeValue = bookShelf.createTextNode("");
16
17   newBookNode.appendChild(newTitleNode);
18   newBookNode.appendChild(newAuthorNode);
19   newTitleNode.appendChild(newTitleNodeValue);
20   newAuthorNode.appendChild(newAuthorNodeValue);
21
22 });

```

Parse an XML String and Log Element Values

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample parses the XML string stored in the `xmlString` variable. The sample selects all config elements in the `xmlDocument` node, loops through them, and logs their contents.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   * @NScriptType Suitelet
4   */
5
6  require(['N/xml'], function(xml) {
7      return {
8          onRequest: function(options) {
9              var xmlString = '<?xml version="1.0" encoding="UTF-8"?><config date="1465467658668" transient="false">Some content</config>';
10
11              var xmlDocument = xml.Parser.fromString({
12                  text: xmlString
13              });
14
15              var bookNode = xml.XPath.select({
16                  node: xmlDocument,
17                  xpath: '//config'
18              });
19
20              for (var i = 0; i < bookNode.length; i++) {
21                  log.debug('Config content', bookNode[i].textContent);
22              }
23          }
24      };
25  });

```

Render an Invoice Into a PDF Using an XML Template

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to render an invoice into a PDF using an XML template in the File Cabinet. This sample requires the Advanced PDF/HTML Templates feature.

Note: This sample script uses the `require` function so that you can copy it into the SuiteScript Debugger and test it. You must use the `define` function in an entry point script (the script you attach to a script record and deploy). For more information, see the help topics [SuiteScript 2.x Script Basics](#) and [SuiteScript 2.x Script Types](#).

```

1  /**
2   * @NApiVersion 2.x
3   *
4   */
5
6  // This sample shows how to render an invoice into a PDF file using an XML template in the file cabinet.
7  // Note that this example requires the Advanced PDF/HTML Templates feature.
8  require(['N/render', 'N/file', 'N/record'],
9      function(render, file, record) {
10         function renderRecordToPdfWithTemplate() {
11             var xmlTemplateFile = file.load('Templates/PDF Templates/invoicePDFTemplate.xml');
12             var renderer = render.create();
13             renderer.templateContent = xmlTemplateFile.getContents();
14             renderer.addRecord('greco', record.load({
15                 type: record.Type.INVOICE,
16                 id: 37
17             }));
18             var invoicePdf = renderer.renderAsPdf();
19         }
20     }
21 );

```



```
19     renderRecordToPdfWithTemplate();  
20     });
```

In the preceding sample, the `invoicePDFTemplate.xml` file was referenced in the File Cabinet. This file is similar to the Standard Invoice PDF/HTML Template found in Customization > Forms > Advanced PDF/HTML Templates.

SuiteScript Use Cases Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This section of the SuiteScript Code Samples Catalog includes the following use case samples:

- [Add a Custom Button to Execute a Suitelet](#)
- [Calculate Commission on a Sales Order](#)
- [Copy a Value to the Item Column](#)
- [Set a Default Posting Period in a Custom Field](#)
- [Track Deposits and Refunds](#)
- [Set the Purchase Order Exchange Rate](#)

Each use case sample is also included in a step-by-step tutorial. For more information, see the help topic [SuiteCloud Customization Tutorials](#).

For module samples available in the SuiteScript Code Samples catalog, see [SuiteScript Samples by Module](#). For plug-in samples, see [Custom Plug-in Samples](#).

Add a Custom Button to Execute a Suitelet

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to add a button to a sales order in Pending Fulfillment status.

For the complete tutorial, see the help topic [Add Custom Button to Execute a Suitelet](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/runtime', 'N/log'], (runtime, log) => {
8      function beforeLoad(scriptContext) {
9          try {
10             const recCurrent = scriptContext.newRecord;
11             const objForm = scriptContext.form;
12             const stStatus = recCurrent.getValue({
13                 fieldId: 'status'
14             });
15             const stSuiteletLinkParam = runtime.getCurrentScript().getParameter({
16                 name: 'custscript_suiteletlink'
17             });
18             const suiteletURL = '\"' + stSuiteletLinkParam + '\"';
19             if (stStatus === 'Pending Fulfillment') {
20                 objForm.addButton({
21                     id: 'custpage_suiteletbutton',
22                     label: 'Open Suitelet',
23                     functionName: 'window.open(' + suiteletURL + ')',
24                 });
25             }
26         } catch(error) {
27             log.error({
28                 title: 'beforeLoad_addButton',
29                 details: error.message
30             });
31         }
32     }
33 }

```

```

32     }
33     return {
34         beforeLoad: beforeLoad
35     };
36 });

```

Calculate Commission on a Sales Order

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to calculate a commission amount and populate that amount to a custom field.

For the complete tutorial, see the help topic [Calculate Commission on Sales Orders](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @ApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/record', 'N/log'], (record, log) => {
8      function afterSubmit(scriptContext) {
9          const stMethodName = 'afterSubmit_calculateCommission';
10         try {
11             if (scriptContext.type !== scriptContext.UserEventType.CREATE && scriptContext.type !== scriptContext.UserEvent
Type.EDIT) {
12                 return;
13             }
14             let stItemType = null;
15             let flMSRPTotalAmt = 0.00;
16             let flMSRPAmt = 0.00;
17             let flNetDistributorCost = 0.00;
18             let flCommissionAmount = null;
19             let flQuantity = 0.00;
20             const recSalesOrder = scriptContext.newRecord;
21             let flSubtotal = parseFloat(recSalesOrder.getValue({
22                 fieldId: 'subtotal'
23             }));
24             const numItems = recSalesOrder.getLineCount({
25                 sublistId: 'item'
26             });
27             for (let intLinenum = 0; intLinenum < numItems; intLinenum++) { flMSRPAmt = parseFloat(recSalesOrder.getSublist
Value({ sublistId: 'item', fieldId: 'custcol_salesorder_msrp', line: intLinenum })); flQuantity = parseFloat(recSalesOr
der.getSublistValue({ sublistId: 'item', fieldId: 'quantity', line: intLinenum })); stItemType = recSalesOrder.getSublist
Value({ sublistId: 'item', fieldId: 'itemtype', line: intLinenum }); if (stItemType !== 'Discount' && stItemType !== 'Subto
tal' && stItemType !== 'Markup') { flMSRPTotalAmt = flMSRPTotalAmt + (flMSRPAmt * flQuantity); } flNetDistributorCost = flMSRPTo
talAmt * 0.5; if (flSubtotal === flNetDistributorCost) { flCommissionAmount = flSubtotal * 0.10; } else if ((flSubtotal > flNetDis
tributorCost) && (flSubtotal <= flMSRPTotalAmt)) { flCommissionAmount = flNetDistributorCost * 0.10 + (flSubtotal - flNetDistribu
torCost) * 0.75; } else { if (flSubtotal > flMSRPTotalAmt) {
28                 flCommissionAmount = flNetDistributorCost * 0.10 + (flMSRPTotalAmt - flNetDistributorCost) * 0.75 + (flSubtotal
- flMSRPTotalAmt) * 0.5;
29             }
30         }
31         const thisSalesOrderID = recSalesOrder.id;
32         const updateSalesOrder = record.load({
33             type: record.Type.SALES_ORDER,
34             id: thisSalesOrderID
35         });
36         updateSalesOrder.setValue({
37             fieldId: 'custbody_commission_amount',
38             value: flCommissionAmount
39         });
40         const updateSalesOrderID = updateSalesOrder.save();

```

```

41     } catch(e) {
42         log.debug({
43             title: stMethodName,
44             details: ' - Exit (Catch)- '
45         });
46         if (e.getDetails !== undefined) {
47             log.error({
48                 title: 'Process Error',
49                 details: JSON.stringify(e)
50             });
51             throw e;
52         } else {
53             log.error({
54                 title: 'Unexpected Error',
55                 details: JSON.stringify(e)
56             });
57             throw error.create({
58                 name: 'Unexpected Error',
59                 message: JSON.stringify(e)
60             });
61         }
62     }
63 }
64 return {
65     afterSubmit: afterSubmit
66 };
67 });

```

Copy a Value to the Item Column

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to copy a value to the item column.

For the complete tutorial, see the help topic [Copy a Value to the Item Column](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType ClientScript
4   * @NModuleScope SameAccount
5   */
6
7  define([], function() {
8      function fieldChanged(context) {
9          try {
10             const recInvoice = context.currentRecord;
11             const stCurrField = context.fieldId;
12             const stCurrSublist = context.sublistId;
13             if (stCurrSublist === 'item' && stCurrField === 'custcol_billingitem') {
14                 let billingitem = recInvoice.getCurrentSublistText({
15                     sublistId: 'item',
16                     fieldId: 'custcol_billingitem'
17                 });
18                 recInvoice.setCurrentSublistText({
19                     sublistId: 'item',
20                     fieldId: 'item',
21                     text: billingitem,
22                 });
23             }
24         } catch(e) {
25             alert(e.name + ': ' + e.message);
26         }
27     }
28     return {
29         fieldChanged: fieldChanged

```

```

30     };
31   });

```

Set a Default Posting Period in a Custom Field

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to set a specific default posting period in a custom transaction field.

For the complete tutorial, see the help topic [Set a Default Posting Period in a Custom Field](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType ClientScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/runtime', 'N/search', 'N/log'], (runtime, search, log) => {
8    function pageInit(context) {
9      try {
10         if (context.mode === 'copy' || context.mode === 'create') {
11           const record = context.currentRecord;
12           const searchOpenAccountingPeriods = search.load({
13             id: 'customsearch_open_accounting_periods'
14           });
15           const resultsFromSearch = searchOpenAccountingPeriods.run().getRange({
16             start: 0,
17             end: 1
18           });
19           record.setValue({
20             fieldId: 'custbody_preferred_posting_period',
21             value: resultsFromSearch[0].id
22           });
23           record.setValue({
24             fieldId: 'postingperiod',
25             value: resultsFromSearch[0].id
26           });
27         }
28       } catch(e) {
29         log.error({
30           title: 'ERROR',
31           details: e
32         });
33       }
34     }
35     function fieldChanged(context) {
36       try {
37         if (context.fieldId === 'custbody_preferred_posting_period') {
38           const record = context.currentRecord;
39           const prefPostPeriod = record.getValue({
40             fieldId: 'custbody_preferred_posting_period'
41           });
42           record.setValue({
43             fieldId: 'postingperiod',
44             value: prefPostPeriod
45           });
46         } else {
47           return;
48         }
49       } catch(e) {
50         log.error({
51           title: 'ERROR',
52           details: e
53         });
54       }
55     }
56   }
57 }

```

```

55     }
56     return {
57         pageInit: pageInit,
58         fieldChanged: fieldChanged
59     };
60 });

```

Track Deposits and Refunds

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to track the balance of deposits and refunds on sales orders.

For the complete tutorial, see the help topic [Track Customer Deposit Balances](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
8      function beforeSubmit(scriptContext) {
9          const contextDep = scriptContext.newRecord;
10         const soID = contextDep.getValue({
11             fieldId: 'salesorder'
12         });
13         if ((soID !== null) && (scriptContext.type === scriptContext.UserEventType.DELETE)) {
14             const depAmt = contextDep.getValue({
15                 fieldId: 'payment'
16             });
17             const salesorder = record.load({
18                 type: record.Type.SALES_ORDER,
19                 id: soID
20             });
21             const status = salesorder.getValue({
22                 fieldId: 'status'
23             });
24             if (status !== 'Billed') {
25                 const soTotalPaid = salesorder.getValue({
26                     fieldId: 'custbody_total_deposit_paid'
27                 });
28                 const soRemainingBalance = salesorder.getValue({
29                     fieldId: 'custbody_balance_remaining'
30                 });
31                 salesorder.setValue({
32                     fieldId: 'custbody_total_deposit_paid',
33                     value: soTotalPaid - depAmt
34                 });
35                 salesorder.setValue({
36                     fieldId: 'custbody_balance_remaining',
37                     value: (soRemainingBalance + depAmt)
38                 });
39                 const id = salesorder.save({
40                     enableSourcing: true,
41                     ignoreMandatoryFields: true
42                 });
43             }
44         }
45     }
46     function afterSubmit(scriptContext) {
47         const contextDep = scriptContext.newRecord;
48         const soID = contextDep.getValue({
49             fieldId: 'salesorder'
50         });

```

```

51     if ((soID !== null) && ((scriptContext.type === scriptContext.UserEventType.CREATE) || (scriptContext.type === scriptContext
t.UserEventType.EDIT))) {
52         const salesorder = record.load({
53             type: record.Type.SALES_ORDER,
54             id: soID
55         });
56         const status = salesorder.getValue({
57             fieldId: 'status'
58         });
59         if (status !== 'Billed') {
60             const soEntity = salesorder.getValue({
61                 fieldId: 'entity'
62             });
63             const soTranId = salesorder.getValue({
64                 fieldId: 'tranid'
65             });
66             const soFullTextTranID = 'Sales Order #' + soTranId;
67             const mySearch = search.load({
68                 id: 'customsearch_sobalancedue'
69             });
70             const entityFilter = search.createFilter({
71                 name: 'name',
72                 operator: search.Operator.IS,
73                 values: soEntity
74             });
75             const soIdFilter = search.createFilter({
76                 name: 'formulatext',
77                 operator: search.Operator.IS,
78                 summary: search.Summary.MAX,
79                 formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application'
then {createdfrom.salesorder} when {type}='Sales Order' then 'Order #' || {number} end",
80                 values: soFullTextTranID
81             });
82             mySearch.filters.push(entityFilter, soIdFilter);
83             const soresults = mySearch.run();
84             mySearch.run().each(function(soresults) {
85                 let soTextID = soresults.getValue({
86                     name: 'formulatext',
87                     summary: search.Summary.GROUP
88                 });
89                 if (soFullTextTranID === soTextID) {
90                     let totalPaid = soresults.getValue({
91                         name: 'formulacurrency',
92                         summary: search.Summary.SUM
93                     });
94                     let soTotal = salesorder.getValue({
95                         fieldId: 'total'
96                     });
97                     let remainingBalanceOnOrder = parseFloat(soTotal) - parseFloat(totalPaid);
98                     salesorder.setValue({
99                         fieldId: 'custbody_total_deposit_paid',
100                        value: totalPaid
101                    });
102                    salesorder.setValue({
103                        fieldId: 'custbody_balance_remaining',
104                        value: remainingBalanceOnOrder
105                    });
106                    let id = salesorder.save({
107                        enableSourcing: true,
108                        ignoreMandatoryFields: true
109                    });
110                }
111            });
112        }
113    }
114 }
115 return {
116     beforeSubmit: beforeSubmit,
117     afterSubmit: afterSubmit
118 };
119 });

```



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
8      function afterSubmit(scriptContext) {
9          const contextDepApp = scriptContext.oldRecord;
10         const depAppId = contextDepApp.id;
11         const soEntity = contextDepApp.getValue({
12             fieldId: 'customer'
13         });
14         const createdFrom = contextDepApp.getValue({
15             fieldId: 'deposit'
16         });
17         const cusDeposit = record.load({
18             type: record.Type.CUSTOMER_DEPOSIT,
19             id: createdFrom,
20             isDynamic: true
21         });
22         const orderId = cusDeposit.getValue({
23             fieldId: 'salesorder'
24         });
25         const soFullTextTranID = cusDeposit.getText({
26             fieldId: 'salesorder',
27         });
28         const mySearch = search.load({
29             id: 'customsearch_sobalancedue'
30         });
31         const entityFilter = search.createFilter({
32             name: 'internalidnumber',
33             operator: search.Operator.IS,
34             values: soEntity
35         });
36         const soIdFilter = search.createFilter({
37             name: 'formulatext',
38             operator: search.Operator.IS,
39             formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then {createdfrom.salesorder} when {type}='Sales Order' then 'Sales Order #' || {number} end",
40             values: soFullTextTranID
41         });
42         mySearch.filters.push(entityFilter, soIdFilter);
43         const soresults = mySearch.run();
44         mySearch.run().each(function(soresults) {
45             let soTextID = soresults.getValue({
46                 name: 'formulatext',
47                 summary: search.Summary.GROUP
48             });
49             if (soFullTextTranID === soTextID) {
50                 let totalPaid = soresults.getValue({
51                     name: 'formulacurrency',
52                     summary: search.Summary.SUM
53                 });
54                 let salesorder = record.load({
55                     type: record.Type.SALES_ORDER,
56                     id: orderId,
57                     isDynamic: true
58                 });
59                 let soTotal = salesorder.getValue({
60                     fieldId: 'total'
61                 });
62                 let remainingBalanceOnOrder = parseFloat(soTotal);
63                 remainingBalanceOnOrder = parseFloat(remainingBalanceOnOrder) - parseFloat(totalPaid);
64                 salesorder.setValue({
65                     fieldId: 'custbody_total_deposit_paid',
66                     value: totalPaid
67                 });

```



```

68         salesorder.setValue({
69             fieldId: 'custbody_balance_remaining',
70             value: remainingBalanceOnOrder
71         });
72         let id = salesorder.save({
73             enableSourcing: true,
74             ignoreMandatoryFields: true
75         });
76     }
77 });
78 }
79 return {
80     afterSubmit: afterSubmit
81 };
82 });

```



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @ApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @ModuleScope SameAccount
5   */
6
7  define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
8      return {
9          afterSubmit: function(scriptContext) {
10             const contextRef = scriptContext.newRecord;
11             const refId = contextRef.id;
12             UpdateSalesOrder(refId);
13         }
14     };
15     function UpdateSalesOrder(refId) {
16         const refund = record.load({
17             type: record.Type.CUSTOMER_REFUND,
18             id: refId,
19             isDynamic: true
20         });
21         const soEntity = refund.getValue({
22             fieldId: 'customer'
23         });
24         const lines = refund.getLineCount({
25             sublistId: 'apply'
26         });
27         for (let i = 0; i < lines; i++) {
28             let depositnum = refund.getSublistText({
29                 sublistId: 'apply',
30                 fieldId: 'internalid',
31                 line: i
32             });
33             let refundamt = refund.getSublistValue({
34                 sublistId: 'apply',
35                 fieldId: 'amount',
36                 line: i
37             });
38             let order = search.lookupFields({
39                 type: search.Type.DEPOSIT_APPLICATION,
40                 id: depositnum,
41                 columns: 'createdfrom.salesorder'
42             });
43             let soFullTextTranID = order['createdfrom.salesorder'][0].text;
44             let orderId = order['createdfrom.salesorder'][0].value;
45             let soTotalPaid = search.lookupFields({
46                 type: search.Type.SALES_ORDER,
47                 id: orderId,
48                 columns: ['total']
49             });
50             let soTotal = soTotalPaid['total'];
51             let mySearch = search.load({

```

```

52         id: 'customsearch_sobalancedue'
53     });
54     let entityFilter = search.createFilter({
55         name: 'internalid',
56         join: 'customer',
57         operator: search.Operator.EQUALTO,
58         summary: search.Summary.MAX,
59         values: soEntity
60     });
61     let soIdFilter = search.createFilter({
62         name: 'formulatext',
63         operator: search.Operator.IS,
64         formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then
{createdfrom.salesorder} when {type}='Sales Order' then 'Sales Order #' || {number} end",
65         values: soFullTextTranID
66     });
67     mySearch.filters.push(entityFilter, soIdFilter);
68     let soresults = mySearch.run();
69     mySearch.run().each(function(soresults) {
70         let soTextID = soresults.getValue({
71             name: 'formulatext',
72             summary: search.Summary.GROUP
73         });
74         if (soFullTextTranID === soTextID) {
75             let totalPaid = soresults.getValue({
76                 name: 'formulacurrency',
77                 summary: search.Summary.SUM
78             });
79             let remainingBalanceOnOrder = parseFloat(soTotal);
80             remainingBalanceOnOrder = parseFloat(remainingBalanceOnOrder) - parseFloat(totalPaid);
81             let salesorder = record.load({
82                 type: record.Type.SALES_ORDER,
83                 id: orderId,
84                 isDynamic: true
85             });
86             salesorder.setValue({
87                 fieldId: 'custbody_total_deposit_paid',
88                 value: totalPaid
89             });
90             salesorder.setValue({
91                 fieldId: 'custbody_balance_remaining',
92                 value: remainingBalanceOnOrder
93             });
94             let id = salesorder.save({
95                 enableSourcing: true,
96                 ignoreMandatoryFields: true
97             });
98         }
99     });
100 }
101 }
102 });

```



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType UserEventScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
8      function afterSubmit(scriptContext) {
9          const contextOrder = scriptContext.newRecord;
10         const soID = contextOrder.id;
11         const salesorder = record.load({
12             type: record.Type.SALES_ORDER,
13             id: soID
14         });

```

```

15     const soTotal = salesorder.getValue({
16         fieldId: 'total'
17     });
18     const soEntity = salesorder.getValue({
19         fieldId: 'entity'
20     });
21     const soTranId = salesorder.getValue({
22         fieldId: 'tranid'
23     });
24     const soFullTextTranID = 'Sales Order #' + soTranId;
25     const mySearch = search.load({
26         id: 'customsearch_sobalancedue'
27     });
28     const entityFilter = search.createFilter({
29         name: 'entity',
30         operator: search.Operator.ANYOF,
31         values: soEntity
32     });
33     const soIdFilter = search.createFilter({
34         name: 'formulatext',
35         operator: search.Operator.IS,
36         formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then {createdfrom.salesorder} when {type}='Sales Order' then 'Sales Order #' || {number} end",
37         values: soFullTextTranID
38     });
39     mySearch.filters.push(entityFilter, soIdFilter);
40     const soresults = mySearch.run();
41     mySearch.run().each(function(soresults) {
42         let soTextID = soresults.getValue({
43             name: 'formulatext',
44             summary: search.Summary.GROUP
45         });
46         if (soFullTextTranID === soTextID) {
47             let totalPaid = soresults.getValue({
48                 name: 'formulacurrency',
49                 summary: search.Summary.SUM
50             });
51             let remainingBalanceOnOrder = parseFloat(parseFloat(soTotal)) - parseFloat(totalPaid);
52             salesorder.setValue({
53                 fieldId: 'custbody_total_deposit_paid',
54                 value: totalPaid
55             });
56             salesorder.setValue({
57                 fieldId: 'custbody_balance_remaining',
58                 value: remainingBalanceOnOrder
59             });
60             let id = salesorder.save({
61                 enableSourcing: true,
62                 ignoreMandatoryFields: true
63             });
64         }
65     });
66 }
67 return{
68     afterSubmit: afterSubmit
69 };
70 });

```

Set the Purchase Order Exchange Rate

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to convert the transaction total on a purchase order to a user-specific currency rather than the currency associated with the vendor.

For the complete tutorial, see the help topic [Set Purchase Order Exchange Rate](#).



Important: This sample uses SuiteScript 2.1. For more information, see the help topic [SuiteScript 2.1](#).

```

1  /**
2   * @NApiVersion 2.1
3   * @NScriptType ClientScript
4   * @NModuleScope SameAccount
5   */
6
7  define(['N/runtime', 'N/currentRecord', 'N/currency', 'N/log'], (runtime, currentRecord, currency, log) => {
8      function saveRecord(context) {
9          try {
10             const stUserCurrency = runtime.getCurrentScript().getParameter({
11                 name: 'custscript_custom_currency'
12             });
13             if (stUserCurrency === " " || stUserCurrency === null || stUserCurrency === undefined) {
14                 throw "Please enter a value for Custom Currency at Home > User Preferences > Custom.";
15             }
16             const purchaseOrder = context.currentRecord;
17             const stTranCurrency = purchaseOrder.getValue({
18                 fieldId: 'currency'
19             });
20             const stTranDate = purchaseOrder.getValue({
21                 fieldId: 'trandate'
22             });
23             const stTotal = purchaseOrder.getValue({
24                 fieldId: 'total'
25             });
26             let flTotalAmount = parseFloat(stTotal);
27             let exchangeRate = currency.exchangeRate({
28                 source: stTranCurrency,
29                 target: stUserCurrency,
30                 date: stTranDate
31             });
32             const flExchangeRate = parseFloat(exchangeRate);
33             const flAmountInUserCurrency = parseFloat(flTotalAmount * flExchangeRate);
34             purchaseOrder.setValue({
35                 fieldId: 'custbody_currency_exchange_rate',
36                 value: flExchangeRate
37             });
38             purchaseOrder.setValue({
39                 fieldId: 'custbody_currency_po_amount',
40                 value: flAmountInUserCurrency
41             });
42         } catch(e) {
43             if (e.getDetails() !== undefined) {
44                 log.error({
45                     title: 'Process Error',
46                     details: JSON.stringify(e)
47                 });
48             } else {
49                 log.error({
50                     title: 'Unexpected Error',
51                     details: JSON.stringify(e)
52                 });
53             }
54             throw (e);
55         }
56         return true;
57     }
58     return {
59         saveRecord: saveRecord
60     };
61 });

```

Custom Plug-in Samples

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This section of the SuiteScript Code Samples Catalog includes the following code samples for custom plug-ins:

- [Create a Custom Plug-in for Inbound E-Document Validation](#)
- [Create a Custom Plug-in Implementation for E-Document Custom Data Source](#)
- [Create a Custom Plug-in Implementation for Sending E-Documents](#)
- [Create a Digital Signature Plug-in Implementation for E-Documents](#)
- [Create a Quality Custom Inspection Rule Plug-in](#)
- [Create a Script for Sending E-Documents](#)
- [Update E-Document Certification Statuses](#)

For more information about custom plug-ins, see the help topic [Custom Plug-in Overview](#).

For module samples available in the SuiteScript Code Samples catalog, see [SuiteScript Samples by Module](#). For use case samples, see [SuiteScript Use Cases Samples](#).

Create a Custom Plug-in for Inbound E-Document Validation

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to create a custom plug-in for inbound e-document validation.

This script can also be found at [Creating a Custom Plug-in for Inbound E-Document Validation](#) as part of the Electronic Invoicing Administrator Guide.

i Note: This sample script uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#) SuiteScript Debugger.

The following code is a sample validation plug-in script.

```

1  /**
2   * Copyright (c) 2017, Oracle and/or its affiliates.
3   *
4   * @NModuleScope public
5   * @NApiVersion 2.x
6   * @NScriptType plugintypeimpl
7   */
8
9  define([], function() {
10     /**
11     * validate - This function is the entry point of our plugin script

```

```

12  * @param {Object} pluginContext
13  * @param {Object} pluginContext.eDocument
14  * @param {String} pluginContext.eDocument.id
15  * @param {String} pluginContext.eDocument.scriptId
16  * @param {String} pluginContext.eDocument.content
17  * @param {Object} pluginContext.eDocument.source
18  * @param {String} pluginContext.eDocument.source.id
19  * @param {String} pluginContext.eDocument.source.text
20  * @param {Object} pluginContext.eDocument.template
21  * @param {String} pluginContext.eDocument.template.id
22  * @param {String} pluginContext.eDocument.template.text
23  * @param {Object} pluginContext.eDocument.status
24  * @param {Integer} pluginContext.eDocument.status.id
25  * @param {String} pluginContext.eDocument.status.text
26  * @param {Object} pluginContext.eDocument.package
27  * @param {String} pluginContext.eDocument.package.id
28  * @param {String} pluginContext.eDocument.package.text
29  * @param {Object} pluginContext.eDocument.transactionType
30  * @param {String} pluginContext.eDocument.transactionType.id
31  * @param {String} pluginContext.eDocument.transactionType.text
32  * @param {Object} pluginContext.eDocument.vendor
33  * @param {String} pluginContext.eDocument.vendor.id
34  * @param {String} pluginContext.eDocument.vendor.text
35  * @returns {Object} result
36  * @returns {Boolean} result.success
37  * @returns {String} result.message
38  */
39  function validate(pluginContext) {
40
41      var eDoc = pluginContext.eDocument;
42      var result = {
43          success: false,
44          message: ""
45      };
46
47
48      // Connect to validation service
49
50
51      // If successful
52      result.success = true;
53      result.message = "Validation successful!";
54
55
56      // Sample result if not successful
57      // result.success = false;
58      // result.message = "Service returned a failed response";
59
60
61      return result;
62  }
63
64
65  return {
66      validate: validate
67  };
68
69  });

```

Create a Custom Plug-in Implementation for E-Document Custom Data Source

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

The following sample shows how to add custom data sources to an e-document template.

For more information about custom plug-in implementations for custom data sources and this script, see the help topic [Creating a Custom Plug-in Implementation for E-Document Custom Data Source](#).

Note: This sample script uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger SuiteScript Debugger](#).

The following code is a sample custom plug-in implementation for e-document custom data source.

```

1  * @NApiVersion 2.x
2  * @NScriptType plugintypeimpl
3  * @NModuleScope Public
4  */
5  define(["N/render"], function(nsrender) {
6      /**
7       * inject - This function will provide the custom data source during the generation process
8       * @param {Object} params
9       * @param {String} params.transactionId
10      * @param {Object} params.transactionRecord
11      * @param {Number} params.userId
12      *
13      * @returns {Object} result
14      * @returns {render.DataSource} result.alias
15      * @returns {string} result.format
16      * @returns {Object | Document | string} result.data
17      */
18
19      function inject(params) {
20          var txnRecord = params.transactionRecord;
21          var txnId = params.transactionId;
22          var userId = params.userId;
23          var customObj = {};
24          log.debug("Custom Object", customObj);
25          return {
26              customDataSources: [
27                  {
28                      format: nsrender.DataSource.OBJECT,
29                      alias: "custom",
30                      data: customObj
31                  }
32              ],
33          };
34      }
35
36      return {
37          inject: inject
38      };
39  });

```

Create a Custom Plug-in Implementation for Sending E-Documents

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This script sample shows how to create a custom plug-in implementation for sending e-documents.

This script can also be found at [Creating a Custom Plug-in Implementation for Sending E-Documents](#) as part of the Electronic Invoicing Administrator Guide.

Note: This sample script uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#).

The following code is a sample custom plug-in implementation for sending e-documents.

```

1  /**
2   * @NApiVersion 2.0
3   * @NScriptType plugintypeimpl
4   */
5   define(["../lib/string_formatter",
6          "../lib/wrapper/ns_wrapper_error",
7          "../lib/wrapper/ns_wrapper_config",
8          "../lib/wrapper/ns_wrapper_email",
9          "../lib/wrapper/ns_wrapper_file",
10         "../app/einvoice/app_einvoice_notifier",
11         "../lib/translator"],
12         function(stringFormatter, error, config, email, file, notifier, translator) {
13
14             /**
15              * send - This function is the entry point of our plugin script
16              * @param {Object} pluginContext
17              * @param {String} pluginContext.scriptId
18              * @param {String} pluginContext.sendMethodId
19              * @param {String} pluginContext.eInvoiceContent
20              * @param {Object} pluginContext.customer
21              * @param {String} pluginContext.customer.id
22              * @param {Array} pluginContext.customer.recipients
23              * @param {Object} pluginContext.transaction
24              * @param {String} pluginContext.transaction.number
25              * @param {String} pluginContext.transaction.id
26              * @param {String} pluginContext.transaction.poNum
27              * @param {Object} pluginContext.sender
28              * @param {String} pluginContext.sender.id
29              * @param {String} pluginContext.sender.name
30              * @param {String} pluginContext.sender.email
31              * @param {Array} pluginContext.attachmentFileIds
32              *
33              * @returns {Object} result
34              * @returns {Boolean} result.success
35              * @returns {String} result.message
36              */
37             function send(pluginContext) {
38
39                 var MSG_NO_EMAIL = translator.getString("ei.sending.sendernoemail");
40                 var MSG_SENT_DETAILS = translator.getString("ei.sending.sentsentdetails");
41
42                 var senderDetails = pluginContext.sender;
43                 var customer = pluginContext.customer;
44                 var transaction = pluginContext.transaction;
45                 var recipientList = customer.recipients;
46                 var result = {};
47                 var parameters;
48                 if (!senderDetails.email) {
49                     parameters = {
50                         EMPLOYEE_NAME: senderDetails.name
51                     };
52                     stringFormatter.setString(MSG_NO_EMAIL);
53                     stringFormatter.replaceParameters(parameters);
54                     result = {
55                         success: false,
56                         message: stringFormatter.toString()
57                     };
58                 } else {
59                     var invoiceSendDetails = {
60                         number: transaction.number,
61                         poNumber: transaction.poNum,
62                         transactionType: transaction.type,
63                         eInvoiceContent: pluginContext.eInvoiceContent,

```



```

64         attachmentFileIds: pluginContext.attachmentFileIds
65     };
66     notifier.notifyRecipient(senderDetails.id, recipientList, invoiceSendDetails);
67
68     parameters = {
69         SENDER: senderDetails.email,
70         RECIPIENTS: recipientList.join(", ")
71     };
72     stringFormatter.setString(MSG_SENT_DETAILS);
73     stringFormatter.replaceParameters(parameters);
74
75     result = {
76         success: true,
77         message: stringFormatter.toString()
78     };
79
80     }
81
82     return result;
83
84     }
85
86     return {
87         send: send
88     };
89 });

```

Create a Digital Signature Plug-in Implementation for E-Documents

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This script sample script shows how to implement a digital signature plug-in.

This script can also be found at [Creating a Digital Signature Plug-in Implementation for E-Documents](#) as part of the Electronic Invoicing Administrator Guide.

i Note: This sample script uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#).

The following code is a sample digital signature plug-in implementation.

```

1  /**
2   * Copyright (c) 2019, Oracle NetSuite and/or its affiliates.
3   *
4   * @NApiVersion 2.x
5   * @NModuleScope Public
6   * @NScriptType plugintypeimpl
7   */
8   define(['N/crypto/certificate', 'N/file'],
9
10      function(certificate, file){
11
12          /**
13           *
14           * @param pluginContext
15           * @param {String} pluginContext.unsignedString
16           * @param {String} pluginContext.subsidiaryId
17           *
18           * @returns {Object} result
19           * @returns {string} result.success
20           * @returns {String} result.signedString
21           * @returns {String} result.message

```

```

22  */
23  function signDocument(pluginContext){
24
25      /**
26       * Extract the values from pluginContext
27       */
28      var unsignedString = pluginContext.unsignedString;
29      var subsidiaryId = pluginContext.subsidiaryId;
30
31      var rootTag = "RootTag";
32      var certificateId = "custcertificatesfd";
33      var algorithm = "SHA1";
34
35      var result = { success : true, signedString : unsignedString, message : "This is default implementation of Digital Signature."};
36
37      /**
38       * Call services to sign the string
39       */
40      try {
41
42          var random = 0;
43          /*var signedXML = certificate.signXml({
44              algorithm : algorithm,
45              certId: certificateId,
46              rootTag : rootTag,
47              xmlString : unsignedString
48          });
49
50          result.success = true;
51          result.signedString = signedXML.asString();
52          result.message = "Document signed successfully";
53      */
54      }catch(e){
55          result.success = false;
56          result.signedString = "";
57          result.message = e.message;
58      }
59
60
61      return result;
62  }
63
64  return {
65      signDocument : signDocument
66  };
67  });

```

Create a Quality Custom Inspection Rule Plug-in

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This script sample script shows how to evaluate quality inspection data and standards to determine if an inspection should pass or fail.

This script can also be found at [Quality Custom Inspection Rule Echo Source](#) as part of Quality Management Administration.

The following code is a sample implementation of a custom inspection rule plug-in.

i Note: This sample script uses the define function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the require function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#).

This implementation simply echos all inputs to the log to assist new developers and returns true.

```

1  /**
2  * @NApiVersion 2.x
3  * @NScriptType plugintypeimpl
4  *
5  * ECHO Implementation of Custom Rule Plug-in for Quality Management SuiteApp
6  */
7  define(['N/log'], function(log) {
8      return {
9          inspectionPassed: function(inspectionObj, fieldObj, otherFieldObjs, standardObjs) {
10              // echo inspectionObj
11              log.debug({
12                  title: 'insepectionObj',
13                  details: 'type:' + inspectionObj.type +
14                      ' name:' + inspectionObj.name +
15                      ' txnId:' + inspectionObj.txnId +
16                      ' itemId:' + inspectionObj.itemId
17              });
18
19              // echo fieldObj
20              log.debug({
21                  title: 'fieldObj',
22                  details: 'name:' + fieldObj.name +
23                      ' value:' + fieldObj.value
24              });
25
26              // echo otherFieldObjs
27              for ( var i in otherFieldObjs ) {
28                  log.debug({
29                      title: 'otherFieldObjs',
30                      details: 'name:' + otherFieldObjs[i].name +
31                          ' value:' + otherFieldObjs[i].value
32                  });
33              }
34
35              // echo standardObjs
36              for ( var i in standardObjs ) {
37                  log.debug({
38                      title: 'standardObjs',
39                      details: 'name:' + standardObjs[i].name +
40                          ' value:' + standardObjs[i].value
41                  });
42              }
43              return true;
44          }
45      }
46  });

```

Create a Script for Sending E-Documents

i Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This script sample script shows how to send e-documents.

This script can also be found at [Creating a Script for Sending E-Documents](#) as part of the Electronic Invoicing Administrator Guide.

i Note: This sample script uses the define function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the require function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#).

The following code is a sample script for sending e-documents.

```

1  /**

```

```

2  * @NApiVersion 2.x
3  * @NModuleScope Public
4  */
5  define(["N/record"], function(record, error) {
6      return {
7          /**
8           * send - Sample implementation: This will copy the e-document content to the document's
9           * Memo field
10          */
11          * @param {Object} plugInContext
12          * @param {String} plugInContext.scriptId
13          * @param {String} plugInContext.sendMethodId
14          * @param {String} plugInContext.eInvoiceContent
15          *
16          * @param {Object} plugInContext.customer
17          * @param {String} plugInContext.customer.id
18          * @param {String[]} plugInContext.customer.recipients
19          *
20          * @param {Object} plugInContext.transaction
21          * @param {String} plugInContext.transaction.id
22          * @param {String} plugInContext.transaction.number
23          * @param {String} plugInContext.transaction.poNum
24          *
25          * @param {Object} plugInContext.sender
26          * @param {String} plugInContext.sender.id
27          * @param {String} plugInContext.sender.name
28          * @param {String} plugInContext.sender.email
29          *
30          *
31          * @returns {Object} result
32          * @returns {Boolean} result.success: determines
33          * @returns {String} result.message: a failure message
34          */
35          send: function(plugInContext) {
36              var result = {
37                  success: true,
38                  message: "Success"
39              };
40              try {
41                  var rec = record.load({
42                      type: record.Type.INVOICE,
43                      id: plugInContext.transaction.id,
44                  });
45                  rec.setValue({
46                      fieldId: "memo",
47                      value: [
48                          "Script ID: " + plugInContext.scriptId,
49                          "Customer: " + plugInContext.customer.name,
50                          "Transaction: " + plugInContext.transaction.number,
51                          "Sender: " + plugInContext.sender.name,
52                          "Recipients: " + plugInContext.customer.recipients.join("\n"),
53                          "Content: " + plugInContext.eInvoiceContent].join("\n\n")
54                      ];
55                  rec.save();
56              } catch (e) {
57                  result.success = false;
58                  result.message = "Failure";
59              }
60              return result;
61          }
62      };
63  });

```

Update E-Document Certification Statuses

Applies to: SuiteScript 2.x | APIs | SuiteCloud Developer

This script sample script shows how to update e-document certification status.

This script can also be found at [Updating E-Document Certification Statuses](#) as part of the Electronic Invoicing Administrator Guide.

Note: This sample script uses the `define` function, which is required for an entry point script (a script you attach to a script record and deploy). You must use the `require` function if you want to copy the script into the SuiteScript Debugger and test it. For more information, see the help topic [SuiteScript Debugger](#).

The following code is a sample script to implement e-document certification status updates.

```

1  /**
2   * Copyright (c) 2017, Oracle and/or its affiliates.
3   *
4   * @NApiVersion 2.x
5   * @NScriptType pluginTypeimpl
6   * @NModuleScope public
7   */
8  define([], function() {
9      /**
10       * send - This function is the entry point of our plugin script
11       * @param {Object} pluginContext
12       * @param {String} pluginContext.scriptId
13       * @param {String} pluginContext.sendMethodId
14       * @param {String} pluginContext.eInvoiceContent
15       * @param {Object} pluginContext.customer
16       * @param {String} pluginContext.customer.id
17       * @param {Array} pluginContext.customer.recipients
18       * @param {Object} pluginContext.transaction
19       * @param {String} pluginContext.transaction.number
20       * @param {String} pluginContext.transaction.id
21       * @param {String} pluginContext.transaction.poNum
22       * @param {Object} pluginContext.sender
23       * @param {String} pluginContext.sender.id
24       * @param {String} pluginContext.sender.name
25       * @param {String} pluginContext.sender.email
26       * @param {Array} pluginContext.attachmentFileIds
27       *
28       * @returns {Object} result
29       * @returns {Boolean} result.success
30       * @returns {String} result.message
31       */
32      return {
33          send: function(pluginContext) {
34              var result = {
35                  success: true,
36                  message: '',
37                  eiStatus: {
38                      "transactionId": pluginContext.transaction.id,
39                      "transactionType": pluginContext.transaction.tranType,
40                      "entity": customer.id,
41                      "eDocStatus": "3",
42                      "eventType": "3",
43                      "details": "The e-Doc successfully certified and is ready for sending.",
44                      "owner": pluginContext.sender.id,
45                      "isUpdateFields": "true",
46                      "extraFieldsForUpdate": {},
47                      "bundleId": "",
48                      "bundleName": ""
49                  }
50              };
51              return result;
52          }
53      }
54  });

```