SuiteScript Code Samples

2023.1

May 10, 2023

Documentation Accessibility

Access to Oracle Support

Sample Code

No Excessive Use of the Service

Beta Features

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

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.

```
/**
 * @NApiVersion 2.x
 */
require(['N/action', 'N/util']function(action, util) {

    // 1a) Bulk execute the specified action on a provided list of record IDs.
    // The params property is an array of parameter objects where each object contains required recordId and arbitrary additional parameters.
    var handle = action.executeBulk({
        recordType: "timebill",
        id: "approve",
        params: [{
                recordId: 1,
                note: "this is a note for 1"
            },
            {
                recordId: 5,
                note: "this is a note for 5"
            },
            {
                recordId: 23,
                note: "this is a note for 23"
            }
        ]
    })
});

// 1b) Bulk execute the specified action on a provided list of record IDs.
// The parameters in the previous sample are similar and can be generated programatically using the map function.
var searchResults = /* result of a search, for example, [1, 5, 23] */ ;
var handle = action.executeBulk({
    recordType: "timebill",
    id: "approve",
    params: searchResults.map(function(v) {
        return {
            recordId: v,
            note: "this is a note for " + v
        };
    })
});

// 2a) Bulk execute the specified action on a provided list of record IDs.
// This time with homogenous parameters, that is, all parameter objects are equal except recordId.
var handle = action.executeBulk({
    recordType: "timebill",
    id: "approve",
    params: searchResults.map(function(v) {
        return {
            recordId: v,
            foo: "bar",
            name: "John Doe"
        };
    })
});

// 2b) Bulk execute the specified action on a provided list of record IDs.
// This time with homogenous parameters. Equivalent to the previous sample.
var commonParams = {
    foo: "bar",
    name: "John Doe"
};
var handle = action.executeBulk({
    recordType: "timebill",
    id: "approve",
    params: searchResults.map(function(v) {
        return util.extend({
            recordId: v
        }, commonParams);
    })
});

// 3) Bulk execute the specified action on a provided list of record IDs.
// This is the simplest usage with no extra parameters besides the record ID.
var handle = action.executeBulk({
    recordType: "timebill",
    id: "approve",
    params: searchResults.map(function(v) {
        return {
            recordId: v
        }
    })
});

// 4) Bulk execute the specified action on all record instances that qualify.
// Since we don't have a list of recordIds in hand, we only provide the callback
// that will later be used to transform a recordId to the corresponding parameters object.
var handle = action.executeBulk({
    recordType: "timebill",
    id: "approve",
    condition: action.ALL_QUALIFIED_INSTANCES,
    paramCallback: function(v) {
        return {
            recordId: v,
            note: "this is a note for " + v
        };
    }
});

// 5) Get a particular action for a particular record type.
var approveTimebill = action.get({
    recordType: "timebill",
    id: "approve"
});

// 6) Bulk execute the previously obtained action on a provided list of record IDs.
// Params are generated the same way as above in action.executeBulk().
var handle = approveTimebill.executeBulk({
    params: searchResults.map(function(v) {
        return {
            recordId: v,
            note: "this is a note for " + v
        };
    })
});

// 7) Bulk execute the previously obtained action on all record instances that qualify.
var handle = approveTimebill.executeBulk({
    condition: action.ALL_QUALIFIED_INSTANCES,
    paramCallback: function(v) {
        return {
            recordId: v,
            note: "this is a note for " + v
        };
    }
});

// 8) Get status of a bulk action execution.
var res = action.getBulkStatus({
    taskId: handle
}); // returns a RecordActionTaskStatus object
log.debug(res.status);
});
```

Find Actions Available for the Timebill Record Asynchronously Using Promise Methods

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.

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType ClientScript
 */
require(['N/action', 'N/record'], function(action, record) {
    // create timebill record
    var rec = record.create({
        type: 'timebill',
        isDynamic: true
    });
    rec.setValue({
        fieldId: 'employee',
        value: 104
    });
    rec.setValue({
        fieldId: 'location',
        value: 312
    });
    rec.setValue({
        fieldId: 'hours',
        value: 5
    });
    var recordId = rec.save();

    // find all qualified actions and then execute approve if available
    action.find.promise({
        recordType: 'timebill',
        recordId: recordId
    }).then(function(actions) {
        console.log("We've got the following actions: " + Object.keys(actions));
        if (actions.approve) {
            actions.approve.promise().then(function(result) {
                console.log("Timebill has been successfully approved");
            });
        } else {
            console.log("The timebill is already approved");
        }
    });
});

// Outputs the following:
// We've got the following actions:
// The timebill has been successfully approved
```

Locate and Execute an Action on a Timebill Record

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

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

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.

```
/**
 * @NApiVersion 2.x
 */
require(['N/action', 'N/record'], function(action, record) {
    // create timebill record
    var rec = record.create({
        type: 'timebill',
        isDynamic: true
    });
    rec.setValue({
        fieldId: 'employee',
        value: 104
    });
    rec.setValue({
        fieldId: 'location',
        value: 312
    });
    rec.setValue({
        fieldId: 'hours',
        value: 5
    });
    var recordId = rec.save();

    var actions = action.find({
        recordType: 'timebill',
        recordId: recordId

    });

    log.debug("We've got the following actions: " + Object.keys(actions));
    if (actions.approve) {
        var result = actions.approve();
        log.debug("Timebill has been successfully approved");
    } else {
        log.debug("The timebill is already approved");
    }
});

// Outputs the following:
// We've got the following actions: approve, reject
// Timebill has been successfully approved
```

N/auth Samples

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

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.

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.

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.

```
/**
 * @NApiVersion 2.x
 */

// This script changes the currently logged-in user's NetSuite email address and password.
require(['N/auth'],function(auth) {
    function changeEmailAndPassword() {
        var password = 'myCurrentPassword';
        auth.changeEmail({
            password: password,
            newEmail: 'auth_test@newemail.com'
        });
        auth.changePassword({
            currentPassword: password,
            newPassword: 'myNewPa55Word'
        });
    }
    changeEmailAndPassword();
});
```

N/cache Samples

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

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.

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.

```
const FOLDER_CACHE_NAME = 'folder_cache';

function folderCacheLoader(context) {
    const PARENT_FOLDER_ID = 0;
    const FOLDER_NAME = 1;
    const folderCacheKey = context.key.split('/');
    const parentFolderId = folderCacheKey[PARENT_FOLDER_ID];
    const folderName = folderCacheKey[FOLDER_NAME];

    var folderId = null;
    search.create ({
        type: search.Type.FOLDER,
        columns: [internalid'],
        filters: [
            ['parent', search.Operator.ANYOF, parentFolderId],
            'AND',
            ['name', search.Operator.IS, folderName]
        ]
    }).run()
    .each(function(folder) {
        folderid = folder.id;
        return false;
    });

    if (!folderId) {
        var folder = record.create({
            type: record.Type.FOLDER
        });
        folder.setValue({
            fieldId: 'parent',
            value: parentFolderId
        });
        folder.setValue({
            fieldId: 'name',
            value: folderName
        });
        folderId = folder.save();
    }
    return folderId;
}

function getFolderCache() {
    return cache.getCache({
        name: FOLDER_CACHE_NAME
    });
}

function folderKey(folderName, parentFolderId) {
    return[parentFolderid, folderName].join('/');
}

function getFolder(folderName, parentFolderId) {
    return getFolderCache().get({
        key: folderKey(folderName, parentFolderId),
        loader: folderCacheLoader
    }):
}
```

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.

```
/**
```

```
 * @NApiVersion 2.1
 * @NScriptType Suitelet
 */

// This script retrieves the name of a city based on a ZIP code from a cache.
define(['N/cache', '/SuiteScripts/zipToCityIndexCacheLoader'], function(cache, lib) {
    const ZIP_CODES_CACHE_NAME = 'ZIP_CODES_CACHE';
    const ZIP_TO_CITY_IDX_JSON = 'ZIP_TO_CITY_IDX_JSON';

    function getZipCodeToCityLookupObj() {
        const zipCache = cache.getCache({
            name: ZIP_CODES_CACHE_NAME
        });
        const zipCacheJson = zipCache.get({
            key: ZIP_TO_CITY_IDX_JSON,
            loader: lib.zipCodeDatabaseLoader
        });
        return JSON.parse(zipCacheJson);
    }

    function findCityByZipCode(options) {
        return getZipCodeToCityLookupObj()[String(options.zip)];
    }

    function onRequest(context) {
        const start = new Date();
        if (context.request.parameters.purgeZipCache === 'true') {
            const zipCache = cache.getCache({
                name: ZIP_CODES_CACHE_NAME
            });
            zipCache.remove({
                key: ZIP_TO_CITY_IDX_JSON
            });
        }
        const cityName = findCityByZipCode({
            zip: context.request.parameters.zipcode
        });

        context.response.writeLine(cityName || 'Unknown :(');

        if (context.request.parameters.auditPerf === 'true') {
            context.response.writeLine('Time Elapsed: ' + (new Date().getTime() - start.getTime()) + ' ms');
        }
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * zipToCityIndexCacheLoader.js
 * @NApiVersion 2.1
 * @NModuleScope Public
 */

//This custom module is a loader function that uses a CSV file to retrieve a value that was missing from a cache.
define(['N/file', 'N/cache'], function(file, cache) {
    const ZIP_CODES_CSV_PATH = '/SuiteScripts/Resources/free-zipcode-CA-database-primary.csv';

    function trimOuterQuotes(str) {
        return (str || '').replace(/^"+/, '').replace(/"+$/, '');
    }

    function zipCodeDatabaseLoader(context) {
        log.audit({
            title: 'Loading Zip Codes',
            details: 'Loading Zip Codes for ZIP_CODES_CACHE'
        });
        const zipCodesCsvText = file.load({
            id: ZIP_CODES_CSV_PATH
        }).getContents();
        const zipToCityIndex = {};
        const csvLines = zipCodesCsvText.split('\n');
        util.each(csvLines.slice(1), function(el) {
            var cells = el.split(',');
            var key = trimOuterQuotes(cells[0]);
            var value = trimOuterQuotes(cells[2]);
            if (parseInt(key, 10))
                zipToCityIndex[String(key)] = value;
        });
        return zipToCityIndex;
    }

    return {
        zipCodeDatabaseLoader: zipCodeDatabaseLoader
    }
});
```

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 the 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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/certificateControl','N/file'],function(cc, file){
    var fileObj = file.load({
        id: 'SuiteScripts/dsa.p12'
    });
    var options = {
        file : fileObj,
        password : '022b490ad4334c7e86a8304f937ec68f',
        name : 'testCert',
        description : 'testDescription',
        scriptId : '_testid',
        subsidiaries : [1,3],
        weekReminder : false,
        monthReminder : true,
        threeMonthsReminder : false
    };
    var newCertificate = cc.createCertificate(options);
    newCertificate.save();

    var loadedCertificate = cc.loadCertificate({
            scriptId : 'custcertificate_testid'
    });
    fileObj = file.load({
        id: 'SuiteScripts/ecdsa.p12'
    });
    loadedCertificate.file = fileObj;
    loadedCertificate.password = '022b490ad4334c7e86a8304f937ec68f';
    loadedCertificate.save();
})
```

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/file', 'N/sftp', 'N/certificateControl'], function(file, sftp, certificateControl) {
    var certPath = 'yyy/certificates'
    var certName = 'apiclient_cert.p12';

// Establish SFTP connection
    log.debug('Establishing secured SFTP connection...');
    var connection = sftp.createConnection({
        username: 'sftpuser',
        keyId: 'custkeysftp_nft_demo_key',
        url: 'my.sftp.example.com',
        port: 22,
        directory: 'inbound',
    hostKey: 'AAAAB3NzaC1yc2EAAAADAQABAAABAQC4gYD1K4lE9QnuYEgRRQChjrAM1+bTT95e71Xv0oQ60ywVQEiedhRqSMbPiCPPB4pjpBdOmPIQCCkug+3XwAQ6uNj3UM11zoGGmg86tyEJT6qGB0SsrQJzHTb3EG38BSrBO0WEzOWeJ8E8YODT3oAj1Nrf8Ls3JbGObRF+0uwJDIllSrFkYS3kWCV27NhBnaytGe7iLBgrJdNVlitNqkxZfK0NsAYCaJWQjQLtz+GFfN5zTbKKNsDa6s/YW7oAMMOI3Q5GQAqdXtKY728WvxYTjr2FsYS/KM6nbq/csTvZHWLE0z2TQtB2H0IIofvEP/QvXwmgEnCeVPcNgRwdHWQf'
    });
  log.debug('Connection established!');
// -----------------------------------------------------------------------

// Create new certificate
  log.debug('Creating certificate...');
  var certScriptId = '_' + (Math.random().toString(36).substring(2, 10));
  var cert = certificateControl.createCertificate();
  cert.name = 'Test Certificate China API';
   cert.description = 'Test Certificate China created using API';
// custcertificate prefix will be added automatically
   cert.scriptId = certScriptId;
   log.debug('Downloading certificate file from SFTP...');
   cert.file = connection.download({
    directory: certPath,
    filename: certName
    });
  log.debug('Successfully downloaded!');
  //guid corresponding to the certificate's password';
  var pwd = '022b490ad4334c7e86a8304f937ec68f',
  cert.password = pwd;
  cert.save();
/**/certScriptId = 'custcertificate' + certScriptId;
log.debug('Certificate "' + cert.name + '" successfuly created with id "' + certScriptId + '"!');
// -----------------------------------------------------------------------
// Rename certificate
  log.debug('Renaming certificate...');
 cert = certificateControl.loadCertificate({scriptId: certScriptId});
  cert.name = 'Test Certificate China API TEMP';
  cert.save();
// Verify new certificate name
/**/cert = certificateControl.loadCertificate({scriptId: certScriptId});
   log.debug('Certificate successfully renamed to "' + cert.name + '"');
// -----------------------------------------------------------------------
// Delete certificate
   log.debug('Deleting certificate "' + cert.name + '"...');
   certificateControl.deleteCertificate(certScriptId);
   log.debug('Certificate deleted!');
// -----------------------------------------------------------------------
// Load the deleted certificate
   log.debug('Attempting to load the deleted certificate to invoke an error...');
   try {
     cert = certificateControl.loadCertificate({scriptId: certScriptId});
     }
  catch (e) {
     log.error(e.message);
     }
})
```

Filter the Digital Certificate List by Subsidiary and File Type

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.

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.

```
/**
 * @NApiVersion 2.x
 */
require(['N/certificateControl'],
      function(certificateControl){
         var all = certificateControl.findCertificates();
         var specificType = certificateControl.findCertificates({
              type: 'PFX'
         });
         var specificSub = certificateControl.findCertificates({
              subsidiary: 93
         });
         var specificTypeAndSub = certificateControl.findCertificates({
               type: 'PFX',
               subsidiary: 93
         });
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/certificateControl','N/https/clientCertificate'],function(cc, cert){
    var yodlee = cc.findCertificates({
        name: 'Yodlee',
        description: 'Yodlee certificate'
    });
    cert.post({certId:yodlee[0].id,url:<url>, body:<body>, headers:<headers>
    });
})
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/certificateControl'], function(cc){
    var usages = cc.findUsages({
        id: 'custcertificate_china',
            operation: cc.Operation.POST
        });
})
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/certificateControl','N/crypto/certificate'], function(cc, certificate){
    var signer = certificate.createSigner({
        certId:'custcertficiate_cert_1',
        algorithm: 'SHA256'
        });
    var result = signer.sign();
    var verifier = certificate.createVerifier({
        certId: 'custcertificate_cert_1',
        algorithm: 'SHA256'
        });
    verifier.update('test');
    verifier.verify(result);
    var res = cc.findUsages();
    ;
})
```

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.

```
/**
* @NApiVersion 2.x
*/
define(['N/commerce/recordView'],
    function (recordView) {
        function service(context) {
            var result = {};
            try {
                result.website= recordView.viewWebsite({
                    id: 2,
                    fields: ["internalid","shiptocountries"]
                });
            }
            catch (e) {
                result.websiteError = e.name + ": " + e.message;
            }

            var options = {
                "ids": [382,388],
                "fields": ["displayname"]
            };
            try {
                result.items= recordView.viewItems(options);
            }
            catch (e) {
                result.itemsError = e.name + " : " + e.message;
            }
            return context.response.write(
                JSON.stringify(result)
            );
        }

        return {
            service: service
        };
    }
);
```

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.

```
require(['N/compress', 'N/file'], function(compress, file) {
var unsavedTxtFile = file.create({
    fileType: 'PLAINTEXT',
    name: 'file.txt',
    contents: 'This is a sample content. This is a sample content. This is a sample content. This is only a sample.'
});

log.debug('#### ORIGINAL FILE #### #');
log.debug('Name: ' + unsavedTxtFile.name);
log.debug('Size: ' + unsavedTxtFile.size + 'b');
log.debug('Contents: ' + unsavedTxtFile.getContents());

log.debug('#### GZIPPED FILE WITH MAX COMPRESSION ####');
var gzippedFile = compress.gzip({
    file: unsavedTxtFile,
    level: 9
});
log.debug('Name: ' + gzippedFile.name);
log.debug('Size: ' + gzippedFile.size + 'b');
log.debug('Contents: ' + gzippedFile.getContents().substring(0, 100));

log.debug('#### GUNZIPPED FILE ####');
var gunzippedFile = compress.gunzip({
    file: gzippedFile
});
log.debug('Name: ' + gunzippedFile.name);
log.debug('File size: ' + gunzippedFile.size + 'b');
log.debug('Contents: ' + gunzippedFile.getContents());

});
});
```

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.

```
require(['N/compress', 'N/file'], function(compress, file) {
    // load/create files to be archived
    var binaryFile = file.load({
        id: 200
    });
    var textFile = file.create({
        name: 'file.txt',
        fileType: 'PLAINTEXT',
        contents: 'This is sample content.'
    });

    // create an archive as a temporary file object
    var archiver = compress.createArchiver();
    archiver.add({
        file: binaryFile
    });
    archiver.add({
        file: textFile,
        directory: 'txt/'
    });
    var zipFile = archiver.archive({
        name: 'myarchive.zip'
    });

    // save the archive to file cabinet
    zipFile.folder = 123;
    zipFile.save();
});
```

N/config Samples

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

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.

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 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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/config'],
    function(config) {
        function setTaxAndEmployerId() {
            var companyInfo = config.load({
                type: config.Type.COMPANY_INFORMATION
            });
            companyInfo.setValue({
                fieldId: 'taxid',
                value: '1122334455'
            });
            companyInfo.setValue({
                fieldId: 'employerid',
                value: '123456789'
            });
            companyInfo.save();
            companyInfo = config.load({
                type: config.Type.COMPANY_INFORMATION
            });
            var taxid = companyInfo.getValue({
                fieldId: 'taxid'
            });
        }
        setTaxAndEmployerId();
    });
```

N/crypto Samples

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

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.

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.

```
/**
 * @NApiVersion 2.1
 */
/**This sample demonstrates the APIs needed to generate a secure key using the SHA512 hashing algorithm.
*  The mySecret variable is a placeholder that must be replaced with a valid secret from your NetSuite account.
*  For information about  secrets management page, see Account Administration > Authentication > Secrets Management
 * in the Help Center.
**/

require(['N/crypto', 'N/encode', 'N/runtime'], (crypto, encode, runtime) => {
    function createSecureKeyWithHash() {
        let mySecret = 'custsecret_my_secret';     //secret id take from secrets management page

        let sKey = crypto.createSecretKey({
            secret: mySecret,
            encoding: encode.Encoding.UTF_8
        });

        let hmacSHA512 = crypto.createHmac({
            algorithm: crypto.HashAlg.SHA512,
            key: sKey
        });

        hmacSHA512.update({
            input: inputString,
            inputEncoding: encode.Encoding.BASE_64
        });
        let digestSHA512 = hmacSHA512.digest({
            outputEncoding: encode.Encoding.HEX
        });
    }
    createSecureKeyWithHash();
});
```

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

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.

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 secret key field is 32 characters. If needed, use the Field.maxLength property to change this value.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget', 'N/runtime', 'N/crypto', 'N/encode'], (ui, runtime, crypto, encode) => {
    function onRequest(option) {
        if (option.request.method === 'GET') {
            let form = ui.createForm({
                title: 'My Credential Form'
            });
            let skField = form.addSecretKeyField({
                id: 'mycredential',
                label: 'Credential',
                restrictToScriptIds: [runtime.getCurrentScript().id],
                restrictToCurrentUser: false
            })
            skField.maxLength = 200;

            form.addSubmitButton();

            option.response.writePage(form);
        } else {
            let form = ui.createForm({
                title: 'My Credential Form'
            });

            const inputString = "YWJjZGVmZwo=";
            let myGuid = option.request.parameters.mycredential;

            // Create the key
            let sKey = crypto.createSecretKey({
                guid: myGuid,
                encoding: encode.Encoding.UTF_8
            });

            try {
                let hmacSha512 = crypto.createHmac({
                    algorithm: 'SHA512',
                    key: sKey
                });
                hmacSha512.update({
                    input: inputString,
                    inputEncoding: encode.Encoding.BASE_64
                });
                let digestSha512 = hmacSha512.digest({
                    outputEncoding: encode.Encoding.HEX
                });
            } catch (e) {
                log.error({
                    title: 'Failed to hash input',
                    details: e
                });
            }

            form.addField({
                id: 'result',
                label: 'Your digested hash value',
                type: 'textarea'
            }).defaultValue = digestSha512;

            option.response.writePage(form);
        }
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * @NApiVersion 2.1
 */

require(['N/crypto/certificate'], (certificate) => {
    let signer = certificate.createSigner({
        certId: 'custcertificate1',
        algorithm: certificate.HashAlg.SHA256
    });
    signer.update('test');

    let result = signer.sign();
    let verifier = certificate.createVerifier({
        certId: 'custcertificate1',
        algorithm: certificate.HashAlg.SHA256
    });
    verifier.update('test');
    verifier.verify(result);
})
```

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.

```
/**
 * @NApiVersion 2.1
 */

require(['N/crypto/certificate','N/file'],(cert, file) => {
    // Load the file from the File Cabinet.
    // Note that the id value is hard-coded in this sample, and you should use
    // a valid file id from your account.
    let infNFe = file.load({
        id: 922
    });
    let signedXml = cert.signXml({
        algorithm: certificate.HashAlg.SHA256,
        certId: 'custcertificate1',
        rootTag: 'infNFe',
        xmlString: infNFe.getContents()
    });
    cert.verifyXMLSignature({
        signedXml:signedXml,
        rootTag: 'infNFe'
    });
});// 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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/currency'], function(currency) {
    function getUSDFromCAD() {
        var canadianAmount = 100;
        var rate = currency.exchangeRate({
            source: 'CAD',
            target: 'USD',
            date: new Date('7/28/2015')
        });

        var usdAmount = canadianAmount * rate;
    }

    getUSDFromCAD();
});
```

N/currentRecord Samples

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/currentRecord'], function(currentRecord){
    var rec = currentRecord.get();
    rec.selectNewLine({
        sublistId: 'item'
    });
    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'item',
        value: 39,
        forceSyncSourcing:true
    });
    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'quantity',
        value: 1,
        forceSyncSourcing:true
    });
    rec.commitLine({sublistId: 'item'});
})
```

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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/currentRecord'], currentRecord => {
    return ({
        test_set_getValue: () => {
            // Get a reference to the currently active record
            let myRecord = currentRecord.get();

            // Set the value of a custom field
            myRecord.setValue({
                fieldId: 'custpage_textfield',
                value: 'Body value',
                ignoreFieldChange: true,
                forceSyncSourcing: true
            });

            // Retrieve the value that was set
            let actValue = myRecord.getValue({
                fieldId: 'custpage_textfield'
            });

            // Set the value of another custom field
            myRecord.setValue({
                fieldId: 'custpage_resultfield',
                value: actValue,
                ignoreFieldChange: true,
                forceSyncSourcing: true
            });
        },

        test_set_getCurrentSublistValue: () => {
            // Get a reference to the currently active record
            let myRecord = currentRecord.get();

            // Set the value of a custom sublist field
            myRecord.setCurrentSublistValue({
                sublistId: 'sitecategory',
                fieldId: 'custpage_subtextfield',
                value: 'Sublist Value',
                ignoreFieldChange: true,
                forceSyncSourcing: true
            });

            // Retrieve the value that was set
            let actValue2 = myRecord.getCurrentSublistValue({
                sublistId: 'sitecategory',
                fieldId: 'custpage_subtextfield'
            });

            // Set the value of another custom field
            myRecord.setValue({
                fieldId: 'custpage_sublist_resultfield',
                value: actValue2,
                ignoreFieldChange: true,
                forceSyncSourcing: true
            });
        }
    });
});
```

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 * @NModuleScope SameAccount
 */
define([], () => {
    return {
        beforeLoad: params => {
            // Get a reference to the current form that is about to load
            let form = params.form;

            // Add several custom fields to the form
            let textfield = form.addField({
                id: 'custpage_textfield',
                type: 'text',
                label: 'Text'
            });
            let resultfield = form.addField({
                id: 'custpage_resultfield',
                type:'text',
                label: 'Result'
            });
            let sublistResultfield = form.addField({
                id: 'custpage_sublist_resultfield',
                type: 'text',
                label: 'Sublist Result Field'
            });

            // Get a reference to the sitecategory sublist
            let sublistObj = form.getSublist({
                id: 'sitecategory'
            });

            // Add a custom field to the sublist
            let subtextfield = sublistObj.addField({
                id: 'custpage_subtextfield',
                type: 'text',
                label: 'Sublist Text Field'
            });

            // Set the module path to the previous sample script
            form.clientScriptModulePath = './clientDemo.js';

            // Add two custom buttons to the form
            form.addButton({
                id: 'custpage_custombutton',
                label: 'SET_GET_VALUE',
                functionName: 'test_set_getValue'
            });
            form.addButton({
                id: 'custpage_custombutton2',
                label: 'SET_GETCURRENTSUBLISTVALUE',
                functionName: 'test_set_getCurrentSublistValue'
            });
        }
    };
});
```

N/dataset Samples

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

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

The folllowing 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.

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.

```
/**
 * @NApiVersion 2.x
 */

// 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.

require(['N/dataset', 'N/query'], function(dataset, query){
    var myTransactionDateColumn = dataset.createColumn({
        fieldId: 'trandate',
        alias: 'date'
    });
    var myJoin = dataset.createJoin({
        fieldId: 'createdby',
        target: 'entity'
    });
    var myNameColumn = dataset.createColumn({
        fieldId: 'lastname',
        alias: 'name',
        join: myJoin
    });
    var myTransactionIdColumn = dataset.createColumn({
        fieldId: 'tranid',
        alias: 'id'
    });
    var myTotalColumn = dataset.createColumn({
        fieldId: 'foreigntotal',
        alias: 'total'
    });
    var myColumns = [myTransactionIdColumn, myNameColumn, myTransactionDateColumn, myTotalColumn];
    var myCondition = dataset.createCondition({
        column: myNameColumn,
        operator: query.Operator.EMPTY
    });
    var myDataset = dataset.create({
        type: 'transaction',
        columns: myColumns,
        condition: myCondition
    });

    // List all created datasets
    var allDatasets = dataset.list();
    log.debug({
        title: 'All datasets:',
        details: allDatasets
    });

    // Review the newly created dataset components (in the log)
    log.debug({
        title: 'My Dataset = ',
        details: myDataset
    });

    // Now run the dataset
    var runResult = myDataset.run();
    var runPagedResult = myDataset.runPaged({
        pageSize: 2
    });
    log.debug({
        title: 'runResult: ',
        details: runResult
    });
    log.debug({
        title: 'runPagedResult: ',
        details: runPagedResult
    });
});
```

List All Datasets and Load the First Dataset

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

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

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.

```
/**
 * @NApiVersion 2.x
 */

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

require(['N/dataset'], function(dataset){
    // List all created datasets
    var allDatasets = dataset.list();
    log.debug({
        title: 'All datasets:',
        details: allDatasets
    });

    // Load the first dataset
    var myFirstDataset = dataset.load({
        id: allDatasets[0].id
    });
    log.debug('myFirstDataset:', myFirstDataset);
});
```

N/email Samples

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

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

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

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: 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.

```
/**
 * @NApiVersion 2.1
 */

// This script sends an email with an attachment.

require(['N/email', 'N/record', 'N/file'], (email, record, file) => {
    const senderId = -515;
    const recipientEmail = 'notify@myCompany.com';
    let timeStamp = new Date().getUTCMilliseconds();

    let recipient = record.create({
        type: record.Type.CUSTOMER,
        isDynamic: true
    });
    recipient.setValue({
        fieldId: 'subsidiary',
        value: '1'
    });
    recipient.setValue({
        fieldId: 'companyname',
        value: 'Test Company' + timeStamp
    });
    recipient.setValue({
        fieldId: 'email',
        value: recipientEmail
    });

    let recipientId = recipient.save();

    let fileObj = file.load({
        id: 88
    });

    email.send({
        author: senderId,
        recipients: recipientId,
        subject: 'Test Sample Email Module',
        body: 'email body',
        attachments: [fileObj],
        relatedRecords: {
            entityId: recipientId,
            customRecord: {
                id: recordId,
                recordType: recordTypeId
            }
        }
    });
});
```

N/encode Samples

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

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

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/encode'], function(encode) {
    function convertStringToDifferentEncoding() {
        var stringInput = "TÃƒÂ©st StriÃƒÂ±g Input";
        var base64EncodedString = encode.convert({
            string: stringInput,
            inputEncoding: encode.Encoding.UTF_8,
            outputEncoding: encode.Encoding.BASE_64
        });
        var hexEncodedString = encode.convert({
            string: stringInput,
            inputEncoding: encode.Encoding.UTF_8,
            outputEncoding: encode.Encoding.HEX
        });
    }

    convertStringToDifferentEncoding();
});
```

N/error Samples

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

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

The following sample shows how to create a custom error.

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.

```
/**
 * @NApiVersion 2.x
 */

// This script creates a custom error.
require(['N/error'], function(error) {
    function createError() {
        var myCustomError = error.create({
            name: 'MY_ERROR_CODE',
            message: 'My custom error details',
            notifyOff: true
        });
    }
    createError();
});
```

Create an Error Based on a Condition

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

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

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.

```
/**
 * @NApiVersion 2.x
 */

// This script conditionally creates and throws an error.
require(['N/error'], function(error) {
    function showError() {
        var someVariable = false;

        if (!someVariable) {
            var myCustomError = error.create({
                name: 'WRONG_PARAMETER_TYPE',
                message: 'Wrong parameter type selected.',
                notifyOff: false
            });

            // This will write 'Error: WRONG_PARAMETER_TYPE Wrong parameter type selected' to the log
            log.error('Error: ' + myCustomError.name , myCustomError.message);
            throw myCustomError;
        }
    }
    showError();
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/file'], file => {
    // Create a file containing text
    // Note that the folder value is hard-coded in this sample, and you should
    // use a valid folder ID from your account
    let fileObj = file.create({
        name: 'testHelloWorld3.txt',
        fileType: file.Type.PLAINTEXT,
        contents: 'Hello World\nHello World',
        folder: -15,
        isOnline: true
    });

    // Save the file
    let id = fileObj.save();

    // Load the same file to ensure it was saved correctly
    fileObj = file.load({
        id: id
    });
});
```

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/file', 'N/error', 'N/log'], function (file, error, log) {
    // This sample calculates the total for the
    // second column value in a CSV file.
    //
    // Each line in the CSV file has the following format:
    // date,amount
    //
    // Here is the data that the script adds to the file:
    // 10/21/14,200.0
    // 10/21/15,210.2
    // 10/21/16,250.3

    // Create the CSV file
    var csvFile = file.create({
        name: 'data.csv',
        contents: 'date,amount\n',
        folder: 39,
        fileType: 'CSV'
    });

    // Add the data
    csvFile.appendLine({
        value: '10/21/14,200.0'
    });
    csvFile.appendLine({
        value: '10/21/15,210.2'
    });
    csvFile.appendLine({
        value: '10/21/16,250.3'
    });

    // Save the file
    var csvFileId = csvFile.save();

    // Create a variable to store the calculated total
    var total = 0.0;

    // Load the file
    var invoiceFile = file.load({
        id: csvFileId
    });

    // Obtain an iterator to process each line in the file
    var iterator = invoiceFile.lines.iterator();

    // Skip the first line, which is the CSV header line
    iterator.each(function () {return false;});

    // Process each line in the file
    iterator.each(function (line) {
        // Update the total based on the line value
        var lineValues = line.value.split(',');
        var lineAmount = parseFloat(lineValues[1]);
        if (!lineAmount) {
            throw error.create({
                name: 'INVALID_INVOICE_FILE',
                message: 'Invoice file contained non-numeric value for total: ' + lineValues[1]
            });

            total += lineAmount;
            return true;
        }
    });

    // At this point, the total is 660.5
    log.debug({
        title: 'total',
        details: total
    });
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/file'], file => {
    // Create a file containing text
    let fileObj = file.create({
        name: 'testHelloWorld.txt',
        fileType: file.Type.PLAINTEXT,
        contents: 'Hello World\nHello World'
    });
```

```

        // Set the folder for the file
        // Note that this value is hard-coded in this sample, and you should use
        // a valid folder ID from your account
        fileObj.folder = -15;

        // Save the file
        let id = fileObj.save();

        // Load the same file to ensure it was saved correctly
        fileObj = file.load({
            id: id
        });
});
```

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

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.

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType bankStatementParserPlugin
 */

define(['N/file', 'N/log'], function(file, log)   {
    return {
        parseBankStatement: function(context) {
            var reader = context.input.file.getReader();

            var textUntilFirstComma = reader.readUntil(',');
            var next10Characters = reader.readChars(10);
            var textUntilNextNewLine = reader.readUntil('\n');
            var next100Characters = reader.readChars(100);

            log.debug({
                title: 'STATEMENT TEXT',
                details: textUntilFirstComma
            });

            log.debug({
                title: 'STATEMENT TEXT',
                details: next10Characters
            });

            log.debug({
                title: 'STATEMENT TEXT',
                details: textUntilNextNewLine
            });

            log.debug({
                title: 'STATEMENT TEXT',
                details: next100Characters
            })
        }
    }
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType bankStatementParserPlugin
 */

define(['N/file', 'N/log'], function(file, log)   {
    return {
        parseBankStatement: function(context) {
            var statementFile = context.input.file;

            var statementSegmentIterator = statementFile.getSegments({separator: '\\|_|/'}).iterator();
            statementSegmentIterator.each(function (segment) {
                log.debug({
                    title: 'STATEMENT TEXT',
                    details: segment.value
                });
                return true;
            });
        }
    }
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/format'],
    function(format){
        function formatToString() {
            // Assume number format is 1.000.000,00 and negative format is (100)
            var rawNum2 =  -44444.44
            return format.format({value:rawNum2, type: format.Type.FLOAT})
            }
        var formattedNum2 = formatToString(); // "44.444,44" -- a string
   });
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/format'],
    function(format){
        function formatTimeOfDay() {
            // Assume the time format is hh:mm (24 hours)
            var now = new Date(); // Say it's 7:01PM right now.
            return format.format({value: now, type: format.Type.TIMEOFDAY})
            }
       var formattedTime = formatTimeOfDay(); // "19:01" -- a string
  });
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget', 'N/format'], function(serverWidget, format) {
    function parseAndFormatDateString() {
        // Assuming Date format is MM/DD/YYYY
        var initialFormattedDateString = "07/28/2015";
        var parsedDateStringAsRawDateObject = format.parse({
            value: initialFormattedDateString,
            type: format.Type.DATE
        });
        var formattedDateString = format.format({
            value: parsedDateStringAsRawDateObject,
            type: format.Type.DATE
        });
        return [parsedDateStringAsRawDateObject, formattedDateString];
    }
    function onRequest(context) {
        var data = parseAndFormatDateString();

        var form = serverWidget.createForm({
            title: "Date"
        });

        var fldDate = form.addField({
            type: serverWidget.FieldType.DATE,
            id: "date",
            label: "Date"
        });
        fldDate.defaultValue = data[0];

        var fldString = form.addField({
            type: serverWidget.FieldType.TEXT,
            id: "dateastext",
            label: "Date as text"
        });
        fldString.defaultValue = data[1];

        context.response.writePage(form);
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
```

```
 * @NApiVersion 2.x
 */

require(['N/format'],
    function(format){
        function parseToValue() {
            // Assume number format is 1.000.000,00 and negative format is -100
            var formattedNum = "-20.000,25"
            return format.parse({value:formattedNum, type: format.Type.FLOAT})
            }
        var rawNum = parseToValue(); // -20000.25 -- a number
   });
```

N/format/i18n Samples

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

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

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

> **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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/format/i18n'],
    function(format) {
        var spellOut = format.spellOut({
            number: 12345,
            locale: "DE"
        });

        log.debug(spellOut);
    });
```

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

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

The following sample formats a number as a 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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/format/i18n'],
    function(format) {
        log.debug("Test of default number formatter:");
        var numberFormatter = format.getNumberFormatter();

        var gs = numberFormatter.groupSeparator;
        log.debug("Group separator: " + gs);

        var ds = numberFormatter.decimalSeparator;
        log.debug("Decimal separator: " + ds);

        var precision = numberFormatter.precision;
        log.debug("Precision: " + precision);

        var nnf = numberFormatter.negativeNumberFormat;
        log.debug("Negative Number Format: " + nnf);

        log.debug(numberFormatter.format({number: 12.53}));
        log.debug(numberFormatter.format({number: 12845.22}));
        log.debug(numberFormatter.format({number: -5421}));
        log.debug(numberFormatter.format({number: 0.00}));
        log.debug(numberFormatter.format({number: 0.3456789}));
    });
```

Format Currency Based on the Locale Parameter

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

The following sample formats currency 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.

```
require(['N/format/i18n'], function(format) {
    var curFormatter = format.getCurrencyFormatter({
        locale: "ar_SA"
    });

    var curCur = curFormatter.currency;
    log.debug("Currency: " + curCur);

    var sym = curFormatter.symbol;
    log.debug("Currency symbol: " + sym);

    var numberFormat = curFormatter.numberFormatter;

    var gs = numberFormat.groupSeparator;
    log.debug("Group separator: " + gs);

    var ds = numberFormat.decimalSeparator;
    log.debug("Decimal separator: " + ds);

    var prec = numberFormat.precision;
    log.debug("Precision: " + prec);

    var nnf = numberFormat.negativeNumberFormat;
    log.debug("Negative Number Format: " + nnf);

    log.debug(curFormatter.format({number: 123456.55}));
    log.debug(curFormatter.format({number: -123456.55}));
});
```

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.

```
require(['N/format/i18n'],
    function(format) {
        var curFormatter = format.getCurrencyFormatter({locale: "en_IN"});

        var curCur = curFormatter.currency;
        log.debug("Currency: " + curCur);

        var numberFormat = curFormatter.numberFormatter;

        var sym = curFormatter.symbol;
        log.debug("Currency symbol: " + sym);

        var gs = numberFormat.groupSeparator;
        log.debug("Group separator: " + gs);

        var ds = numberFormat.decimalSeparator;
        log.debug("Decimal separator: " + ds);

        var prec = numberFormat.precision;
        log.debug("Precision: " + prec);

        var nnf = numberFormat.negativeNumberFormat;
        log.debug("Negative Number Format: " + nnf);

        log.debug(curFormatter.format({number: 12345678.55}));
        log.debug(curFormatter.format({number: -345678.55}));
});
```

Format Numbers as Currency Strings

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

The following sample formats numbers as currency strings.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/format/i18n'],
    function(format) {
                log.debug("Test of currency formatter - EUR:");
        var curFormatter = format.getCurrencyFormatter({currency: "EUR"});

        var curCur = curFormatter.currency;
        log.debug("Currency: " + curCur);

        var numberFormat = curFormatter.numberFormatter;

        var cur3 = curFormatter.symbol;
        log.debug("Currency symbol: " + cur3);

        var c4 = numberFormat.groupSeparator;
        log.debug("Group separator: " + c4);

        var c5 = numberFormat.decimalSeparator;
        log.debug("Decimal separator: " + c5);

        var c6 = numberFormat.precision;
        log.debug("Precision: " + c6);

        var c7 = numberFormat.negativeNumberFormat;
        log.debug("Negative Number Format: " + c7);

        log.debug(curFormatter.format({number: 12.53}));
        log.debug(curFormatter.format({number: -5421}));
        log.debug(curFormatter.format({number: 0.00}));
        log.debug(curFormatter.format({number: 0.3456789}));
    });
```

Format Numbers Based on the Locale Parameter

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.

```
require(['N/format/i18n'], function(format) {
    var numberFormatter = format.getNumberFormatter({locale: "fr_FR"});
    var gs = numberFormatter.groupSeparator;
    log.debug("Group separator: " + gs);

    var ds = numberFormatter.decimalSeparator;
    log.debug("Decimal separator: " + ds);

    var prec = numberFormatter.precision;
    log.debug("Precision: " + prec);

    var nnf = numberFormatter.negativeNumberFormat;
    log.debug("Negative Number Format: " + nnf);

    log.debug(numberFormatter.format({number: 123456.55}));
    log.debug(numberFormatter.format({number: -123456.55}));
});
```

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.

```
require(['N/format/i18n'], function(format) {
    var origNumberStr = "602547854ext.154";
    log.debug("Original number is " + origNumberStr);

    var pnParser = format.getPhoneNumberParser({
        defaultCountry: format.Country.CZECH_REPUBLIC
    });
    var phoneNumber = pnParser.parse({
        number: origNumberStr
    });

    log.debug("Country code: " + phoneNumber.countryCode);
    log.debug("Extension: " + phoneNumber.extension);
    log.debug("National number: " + phoneNumber.nationalNumber);
    log.debug("Number of leading zeros: " + phoneNumber.numberOfLeadingZeros);
    log.debug("Carrier code: " + phoneNumber.carrierCode);
    log.debug("Raw input: " + phoneNumber.rawInput);

    log.debug("\n---------------\nFormatting back:");

    var pnFormatter = format.getPhoneNumberFormatter({});
    var strNumber = pnFormatter.format({
        number: phoneNumber
    });
    log.debug(strNumber);
});
```

Parse a U.S. Phone Number

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

The following sample parses a U.S. phone number and logs the country code, extentions, national number, number of leading zerios, 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.

```
require(['N/format/i18n'], function(format) {
    var origNumberStr = "7524105210ext.154";
    log.debug("Original number is " + origNumberStr);

    var pnParser = format.getPhoneNumberParser({
        defaultCountry: format.Country.UNITED_STATES
    });
    var phoneNumber = pnParser.parse({
        number: origNumberStr
    });

    log.debug(phoneNumber.countryCode);
    log.debug(phoneNumber.extension);
    log.debug(phoneNumber.nationalNumber);
    log.debug(phoneNumber.numberOfLeadingZeros);
    log.debug(phoneNumber.carrierCode);
    log.debug(phoneNumber.rawInput);

    var pnFormatter = format.getPhoneNumberFormatter({
        formatType: format.PhoneNumberFormatType.NATIONAL
    });
    var strNumber = pnFormatter.format({
        number: phoneNumber
    });
    log.debug(strNumber);
});
```

N/http Samples

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

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */

// This script redirects a new sales order record and sets the entity.
define(['N/record', 'N/http'], (record, http)=> {
    function onRequest(context) {
        context.response.sendRedirect({
            type: http.RedirectType.RECORD,
            identifier: record.Type.SALES_ORDER,
            parameters: ({
                entity: 6
            })
        });
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * @NApiVersion 2.1
 */

// This script uses an HTTP GET request for a URL.
require(['N/http'], (http)=> {
    function sendGetRequest() {
        let response = http.get({
            url: 'http://www.google.com'
        });
    }
    sendGetRequest();
});
```

N/https Samples

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

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.

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.

```
/**
 * @NApiVersion 2.1
 */

// This script uses appendSecureString to concatenate strings to use as an API secre.
require(['N/https', 'N/runtime'], (https, runtime) => {
    function concatToCreateSecureString() {
        let baseUrl = https.createSecureString({
            input: 'www.someurl.com/add?apikey='
         });
        let apiKey = https.createSecureString({
            input: '{CUSTSECRET_SOME_INTEGRATION}'
        });
        let url = baseUrl.appendSecureString({
            secureString: apiKey
        });
    }
    concatToCreateSecureString();
});
```

Create a JWT Token Using a Secure String

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType Suitelet
 */

// This script creates a JWT token using https.SecureString.
define(['N/https', 'N/encode'], (https, encode) => {
    function onRequest(context) {
        let nameToken = "custsecret_myName";
        let passwordToken = "custsecret_mypPassword;
        let headerObj = {
            "alg": "HS256",
            "typ": "JWT"
        }
        let payloadObj = {
            "sub": "1234567890",
            "name": "John Doe",
            "iat": 1516239002
        }

        let headerJSON = JSON.stringify(headerObj);
        let payloadJSON = JSON.stringify(payloadObj);
        let headerBASE64 = encode.convert({
            string: headerJSON,
            inputEncoding: encode.Encoding.UTF_8,
            outputEncoding: encode.Encoding.BASE_64_URL_SAFE
        });

        let payloadBASE64 = encode.convert({
            string: payloadJSON,
            inputEncoding: encode.Encoding.UTF_8,
            outputEncoding: encode.Encoding.BASE_64_URL_SAFE
        });

        let headerBASE64 = headerBASE64.replace(/=/g, ""); // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64 without padding
        let payloadBASE64 = payloadBASE64.replace(/=/g, ""); // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64 without padding

        let secStringJwtSignature = https .createSecureString({
            input: headerBASE64 + "." + payloadBASE64
        })
        .hmac({
            algorithm: https.HashAlg.SHA256,
            key: https.createSecretKey({
                      secret: passwordToken,
                      encoding: encode.Encoding.UTF_8
            }),
          resultEncoding: encode.Encoding.BASE_64_URL_SAFE
        })
        .replaceString({ // remove = padding as per JWT spec 'base64UrlEncode' - URL-safe BASE-64 without padding
             pattern: "=",
             replacement: ""
        })

        let secStringJwtAuthHeader = https .createSecureString({
            input: "Bearer " + headerBASE64 + "." + payloadBASE64 + "."
        })
        .appendSecureString({
            secureString: secStringJwtSignature,
            keepEncoding: true
        })

        // Reflect the response using a echo-request suitelet
        let resp = https.get({
            url: "myURL",
            headers: {
                "Authorization": secStringJwtAuthHeader
            }
        });

        {
            log.error("resp-code", resp.code);
            log.debug("resp-body", resp.body);

            let respAuth = JSON.parse(resp.body)["headers"]["Authorization"];

            log.debug("reps-head-auth", respAuth);
            log.debug("reps-head-auth-expected",
                "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.uel3RLILSJ9Q9W2Gomh8vAJQAgdbnd6TS4b7plyFOtA" ); // see https://jwt.io/#debugger-io
        }
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType Suitelet
 */

// This script creates a form with a credential field.
define(['N/ui/serverWidget', 'N/https', 'N/url'], (ui, https, url) => {
    function onRequest(context) {
        if (context.request.method === 'GET') {
            const form = ui.createForm({
                  title: 'Password Form'
            });

            const credField = form.addCredentialField({
                id: 'password',
                label: 'Password',
                restrictToDomains: ['<accountID>.app.netsuite.com'],
                restrictToCurrentUser: false,
                restrictToScriptIds: 'customscript_my_script'
            });

            credField.maxLength = 32;

            form.addSubmitButton();

            context.response.writePage({
                pageObject: form
                });
        }
        else {
            // Request to an existing Suitelet with credentials
            let passwordGuid = context.request.parameters.password;

            // Replace SCRIPTID and DEPLOYMENTID with the internal ID of the suitelet script and deployment in your account
            let baseUrl = url.resolveScript({
                scriptId: SCRIPTID,
                deploymentId: DEPLOYMENTID,
                returnExternalURL: true
            });

            let authUrl = baseUrl + '&pwd={' + passwordGuid + '}';

            let secureStringUrl = https.createSecureString({
                input: authUrl
            });

            let headers = ({
               'pwd': passwordGuid
            });

            let response = https.post({
                credentials: [passwordGuid],
                url: secureStringUrl,
                body: {authorization:' '+ passwordGuid + '', data:'anything can be here'},
                headers: headers
            });
        }
    }
    return {
        onRequest: onRequest
    };
});
```

Create an Authentication Header Using a Secure String

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.

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.

```
/**
```

```
 * @NApiVersion 2.1
 * @NScriptType Suitelet
 */

// This script creates an authentication header using an https.SecureString.
define(['N/https', 'N/encode'], (https, encode) => {
    function onRequest(context) {

        // Secrets with these two Script IDs must be existing and allowed for this script
        const nameToken = "custsecret_myName";
        const passwordToken = "custsecret_mypPassword";

        // Create BASE-64 encoded name:password pair
        const secStringKeyInBase64 = https.createSecureString({
            input: "{" + nameToken + "}:{" + passwordToken + "}"
        });

        secStringKeyInBase64.convertingEncoding({
            toEncoding: encode.Encoding.BASE_64,
            fromEncoding: encode.Encoding.UTF_8
        });

        // Construct the Authorization header
        const secStringBasicAuthHeader = https.createSecureString({
            input: "Basic "
        });

        secStringBasicAuthHeader.appendSecureString({
            secureString: secStringKeyInBase64,
            keepEncoding: true
        });

        // Send the request to third party with the Authorization header
        const resp = https.get({
            url: "myUrl",
            headers: {
                "Authorization": secStringBasicAuthHeader
            }
        });

        // Log the response code
        log.debug("resp-code", resp.code);
    };
    return {
        onRequest: onRequest
    };
});
```

Generate a Secure Token and a Secret Key

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.

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.

```
/**
 * @NApiVersion 2.1
 */

// This script uses a GUID to generate a secure token and a secret key.
require(['N/https', 'N/runtime'], (https, runtime) => {
    function createSecureString() {
        const passwordGuid = '{284CFB2D225B1D76FB94D150207E49DF}';
        let secureToken = https.createSecureString({
            input: passwordGuid
        });
        let secretKey = https.createSecretKey({
            input: passwordGuid
        });
        secureToken = secureToken.hmac({
            algorithm: https.HashAlg.SHA256,
            key: secretKey
        });
    }
    createSecureString();
});
```

N/https/clientCertificate Samples

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

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

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

```
/**
 * @NApiVersion 2.1
 */

require(['N/https/clientCertificate'],(cert)=> {
    // Set the URL
    const url = "https://nfe.fazenda.sp.gov.br/ws/cadconsultacadastro4.asmx";

    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/wsdl/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>";

    const key = "custcertificate1";

    let headers = {
        "Content-Type": "application/soap+xml"
         };

    let response = cert.post({
        url: url,
        certId: key,
        body: data,
        headers: headers
    });
    log.debug(response.body);
})
```

N/keyControl Samples

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

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

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

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget', 'N/file', 'N/keyControl', 'N/runtime'], function(ui, file, keyControl, runtime) {
    function onRequest(context) {
        var request = context.request;
        var response = context.response;

        if (request.method === 'GET') {
            var form = ui.createForm({
                title: 'Enter Password'
            });

            var credField = form.addSecretKeyField({
                id: 'custfield_password',
                label: 'Password',
                restrictToScriptIds: [runtime.getCurrentScript().id],
                restrictToCurrentUser: true //Depends on use case
            });
            credField.maxLength = 64;

            form.addSubmitButton();
            response.writePage(form);
        } else {
            // Read the request parameter matching the field ID we specified in the form
            var passwordToken = request.parameters.custfield_password;

            var pem = file.load({
                id: 422
            });

            var key = keyControl.createKey();
            key.file = pem;
            key.name = 'Test';
            key.password = passwordToken;
            key.save();
        }
    }
    return {
        onRequest: onRequest
    };
});
```

Create a Secret Key

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

The following sample shows how to create a key.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/keyControl','N/file'],function(keyControl,file){
            var key = keyControl.createKey();
            key.file = file.load(422);
        //id of file containing private key (id_ecdsa or id_rsa)
            key.name = "SFTP key";
            key.save();
})
```

N/log Samples

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

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

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

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.

```
/**
```

```
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 */

// This script creates each type of log message.
define(['N/log'],function(log) {
    function beforeLoad(context) {
        var myValue = 'value';

        var myObject = {
            name: 'Jane',
            id: '123'
        };

        // An audit log message
        log.audit({
            title: 'Audit Entry',
            details: myObject
        });

        // A debug log message
        log.debug({
            title: 'Debug Entry',
            details: 'Value of myValue is: ' + myValue
        });

        // An emergency log message
        log.emergency({
            title: 'Emergency Entry',
            details: 'Value of myValue is: ' + myValue
        });

        // An error log message
        log.error({
            title: 'Error Entry',
            details: 'Value of myValue is: ' + myValue
        });
    }
    return {
        beforeLoad: beforeLoad
    };
});
```

N/piremoval Samples

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/piremoval'], function(piremoval) {
    function removePersonalInformation() {
        var piRemovalTask = piremoval.createTask({
            recordType: 'customer',
            recordIds: [11, 19],
            fieldIds: ['comments', 'phone'],
            workflowIds: [1],
            historyOnly: false,
            historyReplacement: 'removed_value'
        });

        piRemovalTask.save();
        var taskId = piRemovalTask.id;

        var piRemovalTaskInProgress = piremoval.loadTask(taskId);
        piRemovalTaskInProgress.run();

        var status = piremoval.getTaskStatus(taskId);
    };

    removePersonalInformation();
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType plugintypeimpl
 */
define(function() {
    return {
        doTheMagic: function(operand1, operand2) {
            return operand1 + operand2;
        }
    }
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/plugin'], function(plugin) {
    function onRequest(context) {
        var impls = plugin.findImplementations({
            type: 'customscript_magic_plugin'
        });

        for (var i = 0; i < impls.length; i++) {
            var pl = plugin.loadImplementation({
                type: 'customscript_magic_plugin',
                implementation: impls[i]
            });
            log.debug('impl ' + impls[i] + ' result = ' + pl.doTheMagic(10, 20));
        }

        var pl = plugin.loadImplementation({
            type: 'customscript_magic_plugin'
        });
        log.debug('default impl result = ' + pl.doTheMagic(10, 20));
    }

    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Portlet
 * @NScriptPortletType form
 */

define([], function() {
    function render(context) {
        var portletObj = context.portlet;
        portletObj.title = 'Test Form Portlet';
        setComponentsForResize();
        setComponentsForRefresh();

        function setComponentsForResize() {
            var DEFAULT_HEIGHT = '50';
            var DEFAULT_WIDTH = '50';
            var inlineHTMLField = portletObj.addField({
                id: 'divfield',
                type: 'inlinehtml',
                label: 'Test inline HTML'
            });
            inlineHTMLField.defaultValue = '<div id=\'divfield_elem\' style=\'border: 1px dotted red; height:
 ' + DEFAULT_HEIGHT + 'px; width: ' + DEFAULT_WIDTH + 'px;\'></div>';
            inlineHTMLField.updateLayoutType({
                layoutType: 'normal'
            });
            inlineHTMLField.updateBreakType({
                breakType: 'startcol'
            });
            var resizeHeight = portletObj.addField({
                id: 'resize_height',
                type: 'text',
                label: 'Resize Height'
            });
            resizeHeight.defaultValue = DEFAULT_HEIGHT;
            var resizeWidth = portletObj.addField({
                id: 'resize_width',
                type: 'text',
                label: 'Resize Width'
            });
            resizeWidth.defaultValue = DEFAULT_WIDTH;
            var resizeLink = portletObj.addField({
                id: 'resize_link',
                type: 'inlinehtml',
                label: 'Resize link'
            });
            resizeLink.defaultValue = resizeLink.defaultValue = '<a id=\'resize_link\' onclick=\"require([\'SuiteScripts/portle
tApiTestHelper\'], function(portletApiTestHelper) {portletApiTestHelper.resizePortlet(); }) \" href=\'#\'>Resize</a><br>';
        }

        function setComponentsForRefresh() {
            var textField = portletObj.addField({
                id: 'refresh_output',
                type: 'text',
                label: 'Date.now().toString()'
            });
            textField.defaultValue = Date.now().toString();
            var refreshLink = portletObj.addField({
                id: 'refresh_link',
                type: 'inlinehtml',
                label: 'Refresh link'
            });
            refreshLink.defaultValue = '<a id=\'refresh_link\' onclick=\'require([\"SuiteScripts/portletApiTestHelper\"], function(portletApiTestHelper) {portletApiTestHelper.refreshPortlet(); }) \' href=\'#\'>Refresh</a>';
        }
    }

    return {
        render: render
    };
})

// portletApiTestHelper.js
define(['N/portlet'], function(portlet) {
    function refreshPortlet() {
        portlet.refresh();
    }

    function resizePortlet() {
        var div = document.getElementById('divfield_elem');
        var newHeight = parseInt(document.getElementById('resize_height').value);
        var newWidth = parseInt(document.getElementById('resize_width').value);
        div.style.height = newHeight + 'px';
        div.style.width = newWidth + 'px';
        portlet.resize();
    }

    return {
        refreshPortlet: refreshPortlet,
        resizePortlet: resizePortlet
    };
});
```

N/query Samples

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

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.

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.

```
/*
 * @NApiVersion 2.x
 */
require(['N/query'], function(query) {
    var customFieldIdQuery = query.create({
        type: query.Type.CUSTOM_FIELD
    });
    customFieldIdQuery.columns = [
        customFieldIdQuery.createColumn({
            fieldId: 'internalid'
        })
    ];
    customFieldIdQuery.condition = customFieldIdQuery.createCondition({
        fieldId: 'scriptid',
        operator: query.Operator.IS,
        values: 'custrecord_my_custom_field'
    });

    var results = customFieldIdQuery.run().asMappedResults();
    var customFieldInternalId = results[0].internalid;
    log.debug(customFieldInternalId);
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/query'], query => {
    // Create a query definition for customer records
    let myCustomerQuery = query.create({
        type: query.Type.CUSTOMER
    });

    // Join the original query definition based on the salesrep field. In a customer
    // record, the salesrep field contains a reference to an employee record. When you
    // join based on this field, you are joining the query definition with the employee
    // query type, and you can access the fields of the joined employee record in
    // your query.
    let mySalesRepJoin = myCustomerQuery.autoJoin({
        fieldId: 'salesrep'
    });

    // Join the joined query definition based on the location field. In an employee
    // record, the location field contains a reference to a location record.
    let myLocationJoin = mySalesRepJoin.autoJoin({
        fieldId: 'location'
    });

    // Create conditions for the query
    let firstCondition = myCustomerQuery.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 107
    });
    let secondCondition = myCustomerQuery.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 2647
    });
    let thirdCondition = mySalesRepJoin.createCondition({
        fieldId: 'email',
        operator: query.Operator.START_WITH_NOT,
        values: 'example'
    });

    // Combine conditions using and() and or() operator methods. In this example,
    // the combined condition states that the id field of the customer record must
    // have a value of either 107 or 2647, and the email field of the employee
    // record (the record that is referenced in the salesrep field of the customer
    // record) must not start with 'example'.
    myCustomerQuery.condition = myCustomerQuery.and(
        thirdCondition, myCustomerQuery.or(firstCondition, secondCondition)
    );

    // Create query columns
    myCustomerQuery.columns = [
        myCustomerQuery.createColumn({
            fieldId: 'entityid'
        }),
        myCustomerQuery.createColumn({
            fieldId: 'id'
        }),
        mySalesRepJoin.createColumn({
            fieldId: 'entityid'
        }),
        mySalesRepJoin.createColumn({
            fieldId: 'email'
        }),
        mySalesRepJoin.createColumn({
            fieldId: 'hiredate'
        }),
        myLocationJoin.createColumn({
            fieldId: 'name'
        })
    ];

    // Sort the query results based on query columns
    myCustomerQuery.sort = [
        myCustomerQuery.createSort({
            column: myCustomerQuery.columns[3]
        }),
        myCustomerQuery.createSort({
            column: myCustomerQuery.columns[0],
            ascending: false
        })
    ];

    // Run the query
    let resultSet = myCustomerQuery.run();

    // Retrieve and log the results
    let results = resultSet.results;
    for (let i = results.length - 1; i >= 0; i--)
        log.debug(results[i].values);
    log.debug(resultSet.types);
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/query'], query => {
    // Create a query definition for transaction records
    let myTransactionQuery = query.create({
        type: query.Type.TRANSACTION
    });

    // Join the original query definition based on the employee field. In a transaction
    // record, the employee field contains a reference to an employee record. When you
    // join based on this field, you are joining the query definition with the employee
    // query type, and you can access the fields of the joined employee record in
    // your query.
    let myEmployeeJoin = myTransactionQuery.autoJoin({
        fieldId: 'employee'
    });

    // Create a condition for the transaction query
    let transactionCondition = myTransactionQuery.createCondition({
        fieldId: 'isreversal',
        operator: query.Operator.IS,
        values: true
    });
    myTransactionQuery.condition = transactionCondition;

    // Create a query column
    myTransactionQuery.columns = [
        myEmployeeJoin.createColumn({
            fieldId: 'subsidiary'
        })
    ];

    // Sort the query results based on a query column
    myTransactionQuery.sort = [
        myTransactionQuery.createSort({
            column: myTransactionQuery.columns[0],
            ascending: false
        })
    ];

    // Run the query as a paged query with 10 results per page
    let results = myTransactionQuery.runPaged({
        pageSize: 10
    });

    log.debug(results.pageRanges.length);
    log.debug(results.count);

    // Retrieve the query results using an iterator
    let iterator = results.iterator();
    iterator.each(function(result) {
        let page = result.value;
        log.debug(page.pageRange.size);
        return true;
    })

    // Alternatively, retrieve the query results by looping through
    // each result
    for (let i = 0; i < results.pageRanges.length; i++)  {
        let page = results.fetch(i);
        log.debug(page.pageRange.size);
    }
});
```

Convert a Query to a SuiteQL and Run It

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.

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.

```
/**
 * @NApiVersion 2.x
 */
require(['N/query'], function(query) {
    var myCustomerQuery = query.create({
        type: query.Type.CUSTOMER
    });

    myCustomerQuery.columns = [
        myCustomerQuery.createColumn({
            fieldId: 'entityid'
        }),
        myCustomerQuery.createColumn({
            fieldId: 'email'
        })
    ];

    myCustomerQuery.condition = myCustomerQuery.createCondition({
        fieldId: 'isperson',
        operator: query.Operator.IS,
        values: [true]
    });

    var mySQLCustomerQuery = myCustomerQuery.toSuiteQL();

    var results = mySQLCustomerQuery.run();
});
```

Create a Query Using a Specific Record Field

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.

```
/**
* @NApiVersion 2.x
*/
require(['N/query'], function(query) {
    var mfgComponent = query.create({
        type: query.Type.MANUFACTURING_COMPONENT
    });

    var mfgOperation = mfgComponent.autoJoin({
        fieldId: 'operationdisplaytext'
    });

    mfgComponent.columns = [
        mfgComponent.createColumn({
            fieldId: 'operationdisplaytext'
        }),
        mfgComponent.createColumn({
            fieldId: 'item'
        }),
        mfgOperation.createColumn({
            fieldId: 'operationsequence'
        })
    ];

    mfgComponent.condition = mfgComponent.and(
        mfgComponent.createCondition({
            fieldId: 'transaction',
            operator: query.Operator.ANY_OF,
            values: <transactionId>
        }),
        mfgOperation.createCondition({
            fieldId: 'operationsequence',
            operator: query.Operator.EQUAL,
            values: 20
        })
    );

    var results = mfgComponent.run();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */
```

```
require(['N/query'], function(query) {
   var sql =
        "SELECT " +
        "  scriptDeployment.primarykey, scriptexecutioncontextmap.executioncontext " +
        " FROM " +
        "  scriptDeployment, scriptexecutioncontextmap " +
        " WHERE " +
        "  scriptexecutioncontextmap.scriptrecord = scriptDeployment.primarykey " +
        " AND " +
        "  scriptexecutioncontextmap.executioncontext IN ('WEBSTORE', 'WEBAPPLICATION')";

    var resultIterator = query.runSuiteQLPaged({
        query: sql,
        pageSize: 10
    }).iterator();

    resultIterator.each(function(page) {
        var pageIterator = page.value.data.iterator();
        pageIterator.each(function(row) {
            log.debug('ID: ' + row.value.getValue(0) + ', Context: ' + row.value.getValue(1));
            return true;
        });
        return true;
    });
});
```

N/record Samples

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

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.

> **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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/record'], function(record) {
    function createPurchaseOrder() {
        var rec = record.create({
            type: 'purchaseorder',
            isDynamic: true
        });
        rec.setValue({
            fieldId: 'entity',
            value: 52
        });
        rec.setValue({
            fieldId: 'location',
            value: 2
        });
        rec.selectNewLine({
            sublistId: 'item'
        });
        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'item',
            value: 190
        });
        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'quantity',
            value: 2
        });
        subrecordInvDetail = rec.getCurrentSublistSubrecord({
            sublistId: 'item',
            fieldId: 'inventorydetail'
        });
        subrecordInvDetail.selectNewLine({
            sublistId: 'inventoryassignment'
        });
        subrecordInvDetail.setCurrentSublistValue({
            sublistId: 'inventoryassignment',
            fieldId: 'receiptinventorynumber',
            value: 'myinventoryNumber'
        });
        subrecordInvDetail.commitLine({
            sublistId: 'inventoryassignment'
        });
        subrecordInvDetail.selectLine({
            sublistId: 'inventoryassignment',
            line: 0
        });
        var myInventoryNumber = subrecordInvDetail.getCurrentSublistValue({
            sublistId: 'inventoryassignment',
            fieldId: 'receiptinventorynumber'
        });
        rec.commitLine({
            sublistId: 'item'
        });
        var recordId = rec.save();
    }
    createPurchaseOrder();
});
```

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/record'], function(record) {
    function createPurchaseOrder() {
        var createRecordPromise = record.create.promise({
            type: 'purchaseorder',
            isDynamic: true
        });
        createRecordPromise.then(function(rec) {
            rec.setValue({
                fieldId: 'entity',
                value: 52
            });
            rec.setValue({
                fieldId: 'location',
                value: 2
            });
            rec.selectNewLine({
                sublistId: 'item'
            });
            rec.setCurrentSublistValue({
                sublistId: 'item',
                fieldId: 'item',
                value: 190
            });
            rec.setCurrentSublistValue({
                sublistId: 'item',
                fieldId: 'quantity',
                value: 2
            });
            subrecordInvDetail = rec.getCurrentSublistSubrecord({
                sublistId: 'item',
                fieldId: 'inventorydetail'
            });
            subrecordInvDetail.selectNewLine({
                sublistId: 'inventoryassignment'
            });
            subrecordInvDetail.setCurrentSublistValue({
                sublistId: 'inventoryassignment',
                fieldId: 'receiptinventorynumber',
                value: 'myinventoryNumber'
            });
            subrecordInvDetail.commitLine({
                sublistId: 'inventoryassignment'
            });
            subrecordInvDetail.selectLine({
                sublistId: 'inventoryassignment',
                line: 0
            });
            var myInventoryNumber = subrecordInvDetail.getCurrentSublistValue({
                sublistId: 'inventoryassignment',
                fieldId: 'receiptinventorynumber'
            });
            rec.commitLine({
                sublistId: 'item'
            });
```

```
                var recordId = rec.save();
        }, function(err) {
            log.error('Unable to create purchase order!', err.name);
        });
    }
    createPurchaseOrder();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

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

     var recordObj = record.create({
         type: record.Type.SALES_ORDER,
         isDynamic: true
     });

     var ENTITY_VALUE = 1;
     var ITEM_VALUE = 1;
     recordObj.setValue({
         fieldId: 'entity',
         value: ENTITY_VALUE
    });
    recordObj.selectNewLine({
         sublistId: 'item'
    });
    recordObj.setCurrentSublistValue({
         sublistId: 'item',
         fieldId: 'item',
         value: ITEM_VALUE
    });
    recordObj.setCurrentSublistValue({
         sublistId: 'item',
         fieldId: 'quantity',
         value: 1
    });
    recordObj.commitLine({
         sublistId:'item'
    });

    var totalBeforeTax = recordObj.getValue({fieldId: 'total'});

    // get macros available on the record
    var macros = recordObj.getMacros();

    // execute the macro
    if ('calculateTax' in macros)
    {
        macros.calculateTax(); // For promise version use: macros.calculateTax.promise()
    }
    // Alternative (direct) macro execution
    // var calculateTax = recordObj.getMacro({id: 'calculateTax'});
    // calculateTax(); // For promise version use: calculateTax.promise()
    var totalAfterTax = recordObj.getValue({fieldId: 'total'});

    var recordId = recordObj.save({
       enableSourcing: false,
       ignoreMandatoryFields: false
    });
});
```

Create and Save a Contact Record

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.

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/record'], record => {
    // Create an object to hold name data for the contact
    const nameData = {
        firstname: 'John',
        middlename: 'Doe',
        lastname: 'Smith'
    };

    // Create a contact record
    let objRecord = record.create({
        type: record.Type.CONTACT,
        isDynamic: true
    });

    // Set the values of the subsidiary, firstname, middlename,
    // and lastname properties
    objRecord.setValue({
        fieldId: 'subsidiary',
        value: '1'
    });
    for (let key in nameData) {
        if (nameData.hasOwnProperty(key)) {
            objRecord.setValue({
                fieldId: key,
                value: nameData[key]
            });
        }
    }

    // Save the record
    let recordId = objRecord.save({
        enableSourcing: false,
        ignoreMandatoryFields: false
    });
});
```

Create and Save a Contact Record Asynchronously Using Promise Methods

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

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

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 */
require(['N/record'], record => {
    // Create an object to hold name data for the contact
    const nameData = {
        firstname: 'John',
        middlename: 'Doe',
        lastname: 'Smith'
    };

    // Create a contact record using the promise method
    let createRecordPromise = record.create.promise({
        type: record.Type.CONTACT,
        isDynamic: true
    });

    // When the promise is fulfilled, set the values of the subsidiary,
    // firstname, middlename, and lastname properties, and save the
    // record
    createRecordPromise.then(objRecord => {
        log.debug('Start evaluating promise content...');
        objRecord.setValue({
            fieldId: 'subsidiary',
            value: '1'
        });
        for (let key in nameData) {
            if (nameData.hasOwnProperty(key)) {
                objRecord.setValue({
                    fieldId: key,
                    value: nameData[key]
                });
            }
        }
        let recordId = objRecord.save({
            enableSourcing: false,
            ignoreMandatoryFields: false
        });
    }, function(e) {
        log.error('Unable to create contact', e.name);
    });
});
```

Create Multiple Sales Records Using a Scheduled Script

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.

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.

```
/**
* @NApiVersion 2.x
* @NScriptType ScheduledScript
*/

// This script creates multiple sales records and logs the record creation progress.
define(['N/runtime', 'N/record'], function(runtime, record) {
    return {
        execute: function(context) {
            var script = runtime.getCurrentScript();
            for (x = 0; x < 500; x++) {
                var rec = record.create({
                    type: record.Type.SALES_ORDER
                });
                script.percentComplete = (x * 100)/500;
                log.debug({
                    title: 'New Sales Orders',
                    details: 'Record creation progress: ' + script.percentComplete + '%'
                });
            }
        }
    };
});
```

N/recordContext Samples

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

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.

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.

```
/**
 *@NApiVersion 2.x
 */
// This example calls the getContext() API with options recordType, recordId and  record.
require(['N/record', 'N/recordContext'],
    function(record, recordContext) {
        // Create record
        var employee = record.create({
            type : record.Type.EMPLOYEE,
            isDynamic: true
        })
        // Setup CA subsidiary
        employee.setValue('subsidiary', 2);
        employee.setValue('entityid', 'test_emp_' + Date.now())
        employeeId = employee.save()

        // getContext() with options recordType, recordId and contextTypes
        var employeeContext = recordContext.getContext({
            recordType : record.Type.EMPLOYEE,
            recordId: employeeId,
            contextTypes: [recordContext.ContextType.LOCALIZATION]
        })
        log.debug(employeeContext);
         // expected log will list {"localization":["CA"]}
         // Change employee subsidiary to AU
         employee.setValue('subsidiary', 3);
        // getContext() with options record and contextTypes
        var employeeContext = recordContext.getContext({
            record : employee,
            contextTypes: [recordContext.ContextType.LOCALIZATION]
        })
        log.debug(employeeContext);
        // expected log will list {"localization":["AU"]}
        // Delete record
        record.delete({
            type : record.Type.EMPLOYEE,
            id: 1
        })
    });
```

N/redirect Samples

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

ORACLE NETSUITE

Set a Redirect URL to a Newly Created Task Record

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.

> **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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/record', 'N/redirect'], function(record, redirect) {
    function redirectToTaskRecord() {
        var taskTitle = 'New Opportunity';
        var taskRecord = record.create({
            type: record.Type.TASK
        });
        taskRecord.setValue('title', taskTitle);

        var taskRecordId = taskRecord.save();

        redirect.toRecord({
            type: record.Type.TASK,
            id: taskRecordId
        });
    }

    redirectToTaskRecord();
});
```

N/render Samples

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/render'],
    function(render) {
        function generatePdfFileFromRawXml() {
            var xmlStr = "<?xml version=\"1.0\"?>\n" +
                "<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\">\n" +
                "<pdf>\n<body font-size=\"18\">\nHello World!\n</body>\n</pdf>";
            var pdfFile = render.xmlToPdf({
                xmlString: xmlStr
            });
        }
        generatePdfFileFromRawXml();
    });
```

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.

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/render'],
    function(render) {
        function renderTransactionToHtml() {
            var transactionFile = render.transaction({
            entityId: 23,
            printMode: render.PrintMode.HTML
            });
        }
        renderTransactionToHtml();
    });
```

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.

```
/**
 * @NApiVersion 2.x
 */

// This sample shows how to render an invoice into a PDF file using an XML template in the file cabinet.
// Note that this example requires the Advanced PDF/HTML Templates feature.
require(['N/render', 'N/file', 'N/record'],
    function(render, file, record) {
        function renderRecordToPdfWithTemplate() {
            var xmlTemplateFile = file.load('Templates/PDF Templates/invoicePDFTemplate.xml');
            var renderer = render.create();
            renderer.templateContent = xmlTemplateFile.getContents();
            renderer.addRecord('grecord', record.load({
                type: record.Type.INVOICE,
                id: 37
            }));
            var invoicePdf = renderer.renderAsPdf();
        }
        renderRecordToPdfWithTemplate();
    });
```

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

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

The following sample shows how to render search results into 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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
// This sample shows how to render search results into a PDF file.
define(['N/render', 'N/search'], function(render, search) {
    function onRequest(options) {
        var request = options.request;
        var response = options.response;

        var xmlStr = '<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n' +
            '<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\">\n' +
            '<pdf lang=\"ru=RU\" xml:lang=\"ru-RU\">\n" + "<head>\n' +
            '<link name=\"russianfont\" type=\"font\" subtype=\"opentype\" ' + 'src=\"NetSuiteFonts/verdana.ttf\" " + "src-bold=\"NetSuiteFonts/verdanab.ttf\"' + 'src-italic=\"NetSuiteFonts/verdanai.ttf\" " + "src-bolditalic=\"NetSuiteFonts/verdanabi.ttf\"' + 'bytes=\"2\"/>\n" + "</head>\n' +
            '<body font-family=\"russianfont\" font-size=\"18\">\n??????? ?????</body>\n" + "</pdf>';

        var rs = search.create({
            type: search.Type.TRANSACTION,
            columns: ['trandate', 'amount', 'entity'],
            filters: []
        }).run();

        var results = rs.getRange(0, 1000);
        var renderer = render.create();
        renderer.templateContent = xmlStr;
        renderer.addSearchResults({
            templateName: 'exampleName',
            searchResult: results
        });

        var newfile = renderer.renderAsPdf();
        response.writeFile(newfile, false);
    }

    return {
        onRequest: onRequest
    };
});
```

N/runtime Samples

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

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.

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.

```
/**
* @NApiVersion 2.x
* @NScriptType ScheduledScript
*/

// This script creates multiple sales records and logs the record creation progress.
define(['N/runtime', 'N/record'], function(runtime, record) {
    return {
        execute: function(context) {
            var script = runtime.getCurrentScript();
            for (x = 0; x < 500; x++) {
                var rec = record.create({
                    type: record.Type.SALES_ORDER
                });
                script.percentComplete = (x * 100)/500;
                log.debug({
                    title: 'New Sales Orders',
                    details: 'Record creation progress: ' + script.percentComplete + '%'
                });
            }
        }
    };
});
```

Return User and Session Information

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.

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */

// This script writes user and session information for the currently executing script to the response.
define(['N/runtime'], function(runtime) {
    function onRequest(context) {
        var remainingUsage = runtime.getCurrentScript().getRemainingUsage();
        var userRole = runtime.getCurrentUser().role;
        var currentSession = runtime.getCurrentSession();

        // Set the current sessions's scope
        currentSession.set({
            name: 'scope',
            value: 'global'
        });

        var sessionScope = runtime.getCurrentSession().get({
            name: 'scope'
        });

        log.debug('Remaining Usage:', remainingUsage);
        log.debug('Role:', userRole);
        log.debug('Session Scope:', sessionScope);

        context.response.write('Executing under role: ' + userRole
            + '. Session scope: ' + sessionScope + '.');
    }
    return {
        onRequest: onRequest
    };
});
```

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`.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    var myCustomRecordSearch = search.create({
        type: search.Type.CUSTOM_RECORD + '6',
        title: 'My Search Title',
        columns: ['custrecord1']
    }).run().each(function(result) {
        // Process each result
        return true;
    });
});
```

Delete a Saved Search

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

The following sample shows how to delete a saved search.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    function deleteSearch() {
        search.delete({
            id: 'customsearch_my_so_search'
        });
    }

    deleteSearch();
});
```

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

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.

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.

```
/**
```

```
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    function runSearchAndFetchResult() {
        var mySearch = search.load({
            id: 'customsearch_my_so_search'
        });

        var searchResult = mySearch.run().getRange({
            start: 0,
            end: 100
        });
        for (var i = 0; i < searchResult.length; i++) {
            var entity = searchResult[i].getValue({
                name: 'entity'
            });
            var subsidiary = searchResult[i].getValue({
                name: 'subsidiary'
            });
        }
    }

    runSearchAndFetchResult();
});
```

Load a Search for Sales Order Records and Use a Callback Function to Process 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 each callback function to process 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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    function loadAndRunSearch() {
        var mySearch = search.load({
            id: 'customsearch_my_so_search'
        });

        mySearch.run().each(function(result) {
            var entity = result.getValue({
                name: 'entity'
            });
            var subsidiary = result.getValue({
                name: 'subsidiary'
            });

            return true;
        });
    }

    loadAndRunSearch();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    function loadAndRunSearch() {
        var mySearch = search.load({
            id: 'customsearch_my_so_search'
        });

        var myPagedData = mySearch.runPaged();
        myPagedData.pageRanges.forEach(function(pageRange){
            var myPage = myPagedData.fetch({index: pageRange.index});
            myPage.data.forEach(function(result){
                var entity = result.getValue({
                    name: 'entity'
                });
                var subsidiary = result.getValue({
                    name: 'subsidiary'
                });
            });
        });
    }

    loadAndRunSearch();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    var mySearch = search.create({
        type: search.Type.CUSTOMER,
        columns: ['entityid', 'firstname', 'lastname', 'salesrep'],
        filters: ['entityid', 'contains', 'Adam']
    });

    var myResultSet = mySearch.run();

    var resultRange = myResultSet.getRange({
        start: 0,
        end: 50
    });

    for (var i = 0; i < resultRange.length; i++) {
        log.debug(resultRange[i]);
    }
});
```

Search for Items in a Custom List

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`.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    var internalId = -1;
    var myCustomListSearch = search.create({
        type: 'customlist_mylist',
        columns: [
            { name : 'internalId' },
            { name : 'abbreviation' }
        ]
    });

    myCustomListSearch.filters = [
        search.createFilter({
            name: 'formulatext',
            formula: '{abbreviation}',
            operator: search.Operator.IS,
            values: abbreviation
        });
    ]

    var resultSet = myCustomListSearch.run();
    var results = resultSet.getRange({
        start: 0,
        end: 1
    });
    for(var i in results) {
        // log.debug('Found custom list record', results[i]);
        internalId = results[i].getValue({
            name:'internalId'
        });
    };
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/search'], function(search) {
    function createSearch() {
        var mySalesOrderSearch = search.create({
            type: search.Type.SALES_ORDER,
            title: 'My SalesOrder Search',
            id: 'customsearch_my_so_search',
            columns: ['entity', 'subsidiary', 'name', 'currency'],
            filters: [
                ['mainline', 'is', 'T'],
                'and', ['subsidiary.name', 'contains', 'CAD']
            ]
        });

        mySalesOrderSearch.save();
    }

    createSearch();
});
```

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.

```javascript
/**
 * @NApiVersion 2.1
 */
require(['N/sftp', 'N/file'], (sftp, file) => {
    // Establish a connection
    log.debug('Establishing SFTP connection...');
    let connection = sftp.createConnection({
        username: 'sftpuser',
        keyId: 'custkeysftp_nft_demo_key',
        url: 'myurl',
        port: 22,
        directory: 'inbound',
        hostKey: 'myhostkey'
    });
    log.debug('Connection established!');

    // List the directory and log the number of elements there
    let list = connection.list({
        path: 'yyy/test'
    });
    log.debug('Items in directory "test" at the beginning: ' + list.length);

    // Generate the test file
    log.debug('Generating test file...');
    let myFileToUpload = file.create({
        name: 'asdf.txt',
        fileType: file.Type.PLAINTEXT,
        contents: 'I am a test file.'
    });
    log.debug('Test file generated, uploading to "test" directory...');

    // Upload the test file
    connection.upload({
        directory: 'yyy/test',
        filename: 'af.txt',
        file: myFileToUpload,
        replaceExisting: true
    });
    log.debug('Upload complete!');

    // List the directory to confirm there is one more file than before
    list = connection.list({
        path: 'yyy/test'
    });
    log.debug('Items in directory "test" after the upload: ' + list.length);

    // Create a new directory
    log.debug('Creating directory "test2"...');
    try {
        connection.makeDirectory({
            path: 'yyy/test2'
        });
        log.debug('Directory created.');
    } catch (e) {
        log.debug('Directory not created.');
        log.error(e.message);
    }
    list = connection.list({
        path: 'yyy/test2'
    });
    log.debug('Items in directory "test2": ' + list.length);

    // Move the test file to the new directory
    log.debug('Moving the test file from "test" to "test2"...');
    connection.move({
        from: 'yyy/test/af.txt',
        to: 'yyy/test2/af.txt'
    })
    log.debug('File moved!');

    // List the original directory again to see the file is moved
    list = connection.list({
        path: 'yyy/test'
    });
    log.debug('Items in directory "test" after the upload: ' + list.length);

    // List the new directory for the file
    list = connection.list({path: 'yyy/test2'});
    log.debug('Items in directory "test2" after the upload: ' + list.length);
    log.debug(JSON.stringify(list));

    // Try to remove the directory
    log.debug('Removing directory "test2"...');
    try {
        connection.removeDirectory({
            path: 'yyy/test2'
        });
        log.debug('Directory removed!');
    } catch (e) {
        log.debug('Directory not removed!');
        log.error(e.message);
    }

    // The directory is not empty so delete the file first
    log.debug('Removing test file from "test2" directory...');
    connection.removeFile({
        path: 'yyy/test2/af.txt'
    });
    log.debug('Test file removed!');

    list = connection.list({
        path: 'yyy/test2'
    });
    log.debug('Items in directory "test2": ' + list.length);

    // Try to remove the directory again
    log.debug('Removing directory "test2"...');
    try {
        connection.removeDirectory({
            path: 'yyy/test2'
        });
        log.debug('Directory removed!');
    } catch (e) {
        log.debug('Directory not removed!');
        log.error(e.message);
    }

    // Try to list the removed directory
    log.debug('Trying to list directory "test2"...');
    try {
        list = connection.list({
            path: 'yyy/test2'
        });
    } catch (e) {
        log.error(e.message);
    }
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 * @NModuleScope SameAccount
 */

define(['N/file', 'N/sftp', 'N/error'],
function(file, sftp, error) {
    return {
        beforeLoad: function(){
            var portNumber = -1;
            var connectTimeout = -1;
            var transferTimeout = -1;
            //these variables can be taken as parameters of the script instead

            if (portNumber <sftp.MIN_PORT_NUMBER || portNumber > sftp.MAX_PORT_NUMBER)
                portNumber = sftp.DEFAULT_PORT_NUMBER;
            if (connectTimeout <sftp.MIN_CONNECT_TIMEOUT) connectTimeout = sftp.MIN_CONNECT_TIMEOUT; else if (connectTimeout >
 sftp.MAX_CONNECT_TIMEOUT)
                connectTimeout = sftp.MAX_CONNECT_TIMEOUT;

            var connection = sftp.createConnection({
                username: 'sftpuser',
                keyId: 'custkey1',
                url: '192.168.0.100',
                port: portNumber,
                directory: 'inbound',
                timeout: connectTimeout,
                hostKey: "AAAAB3NzaC1yc2EAAAADAQABAAABAQDMifKH2vTxdiype8nem7+lS3x7dTQR/A67KdsR/5C2WUcDipBzYhHb
nG6Am12Nd2tlM01LnaBZA6/8P4Y9x/sGTxtsdE/MzeGDUBn6HBlQvgIrhX62wgoKGQ+P2lEAO1+Vz8y3/MB1NmD7Fc62cJ9Mu88YA6jwJOIPZeHYNVyIm9OrY6VyzYyvSJh
H0x7SXyvGnijJQF4G8C4c8u/UVpF/sE16xKZtly2Rx0aDL2FsDRtpyPmM602/R6ISbsmgab3MzzAEIu+zLDMdIBJn3cDhNt1F7Rar6Tu0u18KCkk8GPxbnxDuG4sCNOnX
PYkDXSMUbM/ocRjYGtqdZUMmeTf3"
            });

            // can also be a big file (created for example by async search)
            var myFileToUpload = file.create({
                name: 'originalname.txt',
                fileType: file.Type.PLAINTEXT,
                contents: 'I am a test file.'
            });

            if (myFileToUpload.size > connection.MAX_FILE_SIZE)
                throw error.create({name:"FILE_IS_TOO_BIG", message:"The file you are trying to upload is too big"});

            var minTransferTimeout = 10;
            if (transferTimeout > connection.MAX_TRANSFER_TIMEOUT)
                transferTimeout = connection.MAX_TRANSFER_TIMEOUT;
            else if (transferTimeout < minTransferTimeout)
                transferTimeout = minTransferTimteout;

            connection.upload({
                directory: 'files',
                filename: 'test.txt',
                file: myFileToUpload,
                replaceExisting: true,
                timeout: transferTimeout
            });
        }
    };
});
```

Upload and Download a File

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.

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/sftp', 'N/file'], (sftp, file) => {
    const myPwdGuid = "B34672495064525E5D65032D63B52301";
    const myHostKey = "AAA1234567890Q=";

    // Establish a connection to a remote FTP server
    let connection = sftp.createConnection({
        username: 'myuser',
        passwordGuid: myPwdGuid,
        url: 'host.somewhere.com',
        directory: 'myuser/wheres/my/file',
        hostKey: myHostKey
    });

    // Create a file to upload using the N/file module
    let myFileToUpload = file.create({
        name: 'originalname.js',
        fileType: file.Type.PLAINTEXT,
        contents: 'I am a test file.'
    });

    // Upload the file to the remote server
    connection.upload({
        directory: 'relative/path/to/remote/dir',
        filename: 'newFileNameOnServer.js',
        file: myFileToUpload,
        replaceExisting: true
    });

    // Download the file from the remote server
    let downloadedFile = connection.download({
        directory: 'relative/path/to/file',
        filename: 'downloadMe.js'
    });
});
```

N/sso Samples

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

// This script generates a new OAuth oken for a user.
require(['N/sso'], function(sso) {
    function generateSSOToken() {
        var suiteSignOnRecordId = 1;
        var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
    }
    generateSSOToken();
});
```

Generate a suiteSignOn Token in a Portlet

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Portlet
 * @NScriptPortletType form
 */

// This script uses generateSuiteSignOnToken in a portlet.
define(['N/sso'], function (sso) {
    function render(context) {
        var suiteSignOnRecordId = 'customsso_test';
        var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
        log.debug(url);
    }
    return {
        render: render
    };
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */

// This script uses generateSuiteSignOnToken in a Suitelet.
```

```
define(['N/sso'], function(sso) {
    function onRequest(context) {
        var suiteSignOnRecordId = 'customsso_test'; //Replace placeholder values
        var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
        log.debug(url);
    }
    return {
        onRequest: onRequest
    };
});
```

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 * @NModuleScope SameAccount
 */

// This script uses generateSuiteSignOnToken in a user event script.
define(['N/sso'], function(sso) {
    function beforeLoad(context) {
        var suiteSignOnRecordId = 'customsso_test';
        var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
        log.debug(url);
    }
    return {
        beforeLoad: beforeLoad
    };
});
```

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

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.

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.

```
/*
 * @NApiVersion 2.x
 *
 */

// 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.

require(['N/suiteAppInfo'], function (suiteAppInfo){
    var isBundleInstalled = suiteAppInfo.isBundleInstalled({
        bundleId: '1234'
    });

   var isSuiteAppInstalled = suiteAppInfo.isSuiteAppInstalled({
        suiteAppId: '789'
    });

    var allMyBundlesInstalled = suiteAppInfo.listInstalledBundles();

    var allMySuiteAppsInstalled = suiteAppInfo.listInstalledSuiteApps();

    var myScripts = {"scriptA", "scriptB", "scriptC"};
    var scriptInBundles = suiteAppInfo.listBundlesContainingScripts({
        scriptIds: myScripts
    });

    var scriptInSuiteApps = suiteAppInfo.listSuiteAppsContainingScripts({
        scriptsIds: myScripts
    });
})
```

N/task Samples

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

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.

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/task', 'N/runtime', 'N/email'], (task, runtime, email) => {
    // Store the script ID of the script to submit
    //
    // Update the following statement so it uses the script ID
    // of the map/reduce script record you want to submit
    const mapReduceScriptId = 'customscript_test_mapreduce_script';

    // Create a map/reduce task
    //
    // Update the deploymentId parameter to use the script ID of
    // the deployment record for your map/reduce script
    let mrTask = task.create({
        taskType: task.TaskType.MAP_REDUCE,
        scriptId: mapReduceScriptId,
        deploymentId: 'customdeploy_test_mapreduce_script'
    });

    // Submit the map/reduce task
    let mrTaskId = mrTask.submit();

    // Check the status of the task, and send an email if the
    // task has a status of FAILED
    //
    // Update the authorId value with the internal ID of the user
    // who is the email sender. Update the recipientEmail value
    // with the email address of the recipient.
    let taskStatus = task.checkStatus(mrTaskId);
    if (taskStatus.status === 'FAILED') {
        const authorId = -5;
        const recipientEmail = 'notify@myCompany.com';
        email.send({
            author: authorId,
            recipients: recipientEmail,
            subject: 'Failure executing map/reduce job!',
            body: 'Map reduce task: ' + mapReduceScriptId + ' has failed.'
        });
    }
});
```

Create and Submit a Task with Dependent Scripts

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/task'], function(task) {
    // Specify a file for the search results
    var asyncSearchResultFile = 'SuiteScripts/ExportFile.csv';

    // Create a scheduled script task
    var scheduledScript = task.create({
        taskType: task.TaskType.SCHEDULED_SCRIPT
    });
    scheduledScript.scriptId = 'customscript_as_ftr_ss';
    scheduledScript.deploymentId = 'customdeploy_ss_dpl';
    scheduledScript.params = {
        'custscript_ss_as_srch_res' : asyncSearchResultFile
    };

    // Create a map/reduce script task
    var mapReduceScript = task.create({
        taskType: task.TaskType.MAP_REDUCE
    });
    mapReduceScript.scriptId = 'customscript_as_ftr_mr';
    mapReduceScript.deploymentId = 'customdeploy_mr_dpl';
    mapReduceScript.params = {
        'custscript_mr_as_srch_res' : asyncSearchResultFile
    };

    // Create the search task
    var asyncTask = task.create({
        taskType: task.TaskType.SEARCH
    };
    asyncTask.savedSearchId = 'customsearch35';
```

```javascript
    asyncTask.filePath = asyncSearchResultFile;

    // Add dependent scripts to the search task before it is submitted
    asyncTask.addInboundDependency(scheduledScript);
    asyncTask.addInboundDependency(mapReduceScript);

    // Submit the search task
    var asyncTaskId = asyncTask.submit();
});
```

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

```javascript
/**
 * @NApiVersion 2.x
 * @NScriptType ScheduledScript
 */
define(['N/file', 'N/log', 'N/email', 'N/runtime'], function(file, log, email, runtime) {
    // Load the search results file and send an email with the file attached and
    // the number of rows in the file

    function execute(context) {
        // Read a CSV file and return the number of rows minus the header row
        function numberOfRows(csvFileId) {
            var invoiceFile = file.load({
                id: csvFileId
            });
            var iterator = invoiceFile.lines.iterator();
            var noOfLines = 0;

            // Skip the first row (the header row)
            iterator.each(function() {
                return false;
            });

            // Process the rest of the rows
            iterator.each(function() {
                noOfLines++;
                return true;
            });

            return noOfLines;
        }

        // Send an email to the user who ran the script, and attach the
        // CSV file with the search results
        function sendEmailWithAttachment(csvFileId) {
            var noOfRows = numberOfRows(csvFileId);
            var userId = runtime.getCurrentUser().id;
            var fileObj = file.load({
                id: csvFileId
            });

            email.send({
                author: userId,
                recipients: userId,
                subject: 'Search completed',
                body: 'CSV file attached, ' + noOfRows + ' record(s) found.',
                attachments: [fileObj]
            });
        }

        // Retrieve the ID of the search results file
        //
        // Update the name parameter to use the script ID of the original
        // search task
        var resFileId = runtime.getCurrentScript().getParameter({
            name: 'custscript_ss_as_srch_res'
        });

        if (!resFileId) {
            log.error('Could not obtain file content from the specified ID.');
            return;
        }

        log.debug({
            title: 'search - numberOfRows',
            details: numberOfRows(resFileId)
        });
        sendEmailWithAttachment(resFileId);
    }

    return {
        execute: execute
    };
});
```

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

```
/**
 * @NApiVersion 2.x
 * @NScriptType MapReduceScript
 * @NModuleScope SameAccount
 */
define(['N/runtime', 'N/file', 'N/log', 'N/email'], function(runtime, file, log, email) {
    // Load the search results file, count the number of letters in the file, and
    // store this count in another file

    function getInputData() {
        // Retrieve the ID of the search results file
        //
        // Update the completionScriptParameterName value to use the script
        // ID of the original search task
        var completionScriptParameterName = 'custscript_mr_as_srch_res';
        var resFileId = runtime.getCurrentScript().getParameter({
            name: completionScriptParameterName
        });

        if (!resFileId) {
            log.error({
                details: 'resFileId is not valid. Please check the script parameter stored in the completionScriptParameterName
 variable in getInputData().'
            });
        }

        return {
            type: 'file',
            id: resFileId
        };
    }

    function map(context) {
        var email = context.value.split(',')[1];
        if ("Email" !== email) {
            var splitEmail = email.split('@');
            context.write(splitEmail[splitEmail.length-1], 1);
        }
    }

    function reduce(context) {
        context.write(context.key, context.values.length);
    }

    function summarize(summary) {
        var type = summary.toString();
        log.audit({title: type + ' Usage Consumed ', details: summary.usage});
        log.audit({title: type + ' Concurrency Number ', details: summary.concurrency});
        log.audit({title: type + ' Number of Yields ', details: summary.yields});

        var contents = '';
        summary.output.iterator().each(function(key, value) {
            contents += (key + ' ' + value + '\n');
            return true;
        });

        // Create the output file
        //
        // Update the name parameter to use the file name of the output file
        var fileObj = file.create({
            name: 'domainCount.txt',
            fileType: file.Type.PLAINTEXT,
            contents: contents
        });

        // Specify the folder location of the output file, and save the file
        //
        // Update the fileObj.folder property with the ID of the folder in
        // the file cabinet that contains the output file
        fileObj.folder = -15;
        fileObj.save();
    }

    return {
        getInputData: getInputData,
        map: map,
        reduce: reduce,
        summarize: summarize
    };
});
```

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

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.

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.

```
/**
 * @NApiVersion 2.1
 */
require(['N/task'], task => {
    // Do one of the following:
    //
    // - Create a saved search and capture its ID. To do this, you can use
    //   the following code snippet (replacing the type, id, filters, and
    //   columns values as appropriate):
    //
    //   let mySearch = search.create({
    //       type: search.Type.SALES_ORDER,
    //       id: 'customsearch_my_search',
    //      filters: [...],
    //      columns: [...]
    //   });
    //   mySearch.save();
    //   let savedSearchId = mySearch.searchId;
    //
    // - Use the ID of an existing saved search. This is the approach that
    //   this script sample uses. Update the following statement with the
    //   internal ID of the search you want to use.
    const savedSearchId = 669;

    // Create the search task
    let myTask = task.create({
        taskType: task.TaskType.SEARCH
    });
    myTask.savedSearchId = savedSearchId;

    // Specify the ID of the file that search results will be exported into
    //
    // Update the following statement so it uses the internal ID of the file
    // you want to use
    myTask.fileId = 448;

    // Submit the search task
    let myTaskId = myTask.submit();

    // Retrieve the status of the search task
    let taskStatus = task.checkStatus({
        taskId: myTaskId
    });

    // Optionally, add logic that executes when the task is complete
    if (taskStatus.status === task.TaskStatus.COMPLETE) {
        // Add any code that is appropriate. For example, if this script created
        // a saved search, you may want to delete it.
    }
});
```

Submit a Record Action Task and Check Status

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/task'], function(task) {
    var recordActionTask = task.create({
        taskType: task.TaskType.RECORD_ACTION
    });
    recordActionTask.recordType = 'timebill';
    recordActionTask.action = 'approve';
    recordActionTask.params = [
        {recordId: 1, note: 'This is a note for 1'},
        {recordId: 5, note: 'This is a note for 5'},
        {recordId: 23, note: 'This is a note for 23'}
    ];

    var handle = recordActionTask.submit();

    var res = task.checkStatus({
        taskId: handle
    });    // Returns a RecordActionTaskStatus object
    log.debug('Initial status: ' + res.status);
});
```

N/task/accounting/recognition Samples

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/task/accounting/recognition', 'N/search'], function(recognition, search){
    var mySearch = search.load({
        id: 'customsearch22'
    });

    var elementsList = [];
    mySearch.run().each(function(result) {
        var id = result.getValue({
            name: 'internalid'
        });
        elementsList.push(id);
    });

    var recognitionTask = recognition.create({
        taskType: recognition.TaskType.MERGE_ARRANGEMENTS_TASK
    });

    recognitionTask.arrangements = elementsList;
    recognitionTask.revenueArrangementDate = new Date(2019, 2, 10);

    var taskStatusId = recognitionTask.submit();
    log.debug('taskId = ' + taskStatusId);

    var mergeTaskState = recognition.checkStatus({
        taskId: taskStatusId
    });

    log.debug('Submission ID = ' + mergeTaskState.submissionId);
    log.debug('Resulting Arrangement ID = ' + mergeTaskState.resultingArrangement);
    log.debug('status = ' + mergeTaskState.status);
    log.debug('Error message = ' + mergeTaskState.errorMessage);
});
```

Merge Revenue Arrangements Using an Ad-Hoc Search

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/task/accounting/recognition', 'N/search'], function(recognition, search) {
    var elementsList = [];
    var rs = search.create({
        type: 'revenueelement',
        columns: [
            'internalid'
        ]
    }).run();

    var results = rs.getRange(0, 50);
    for (var i = 0; i < results.length; i++) {
        var id = result.getValue('elementsList');
        elementsList.push(id);
    }

    var t = recognition.create({
        taskType: recognition.TaskType.MERGE_ELEMENTS_TASK
    });
    t.elements = elementsList;
    t.revenueArrangementDate = new Date(2019, 1, 1);

    var taskId = t.submit();
    log.debug('Initial status: ' + res.status);
});
```

Merge Revenue Elements Using Internal IDs

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/task/accounting/recognition'], function(recognition){
    var elementsList = [];
    elementsList.push(401);
    elementsList.push(402);

    var recognitionTask = recognition.create({
        taskType: recognition.TaskType.MERGE_ELEMENTS_TASK
    });
    recognitionTask.elements = elementsList;
    var taskStatusId = recognitionTask.submit();

    var mergeTaskState = recognition.checkStatus({
        taskId: taskStatusId
    });

    log.debug('Submission ID = ' + mergeTaskState.submissionId);
    log.debug('Resulting Arrangement ID = ' + mergeTaskState.resultingArrangement);
    log.debug('status = ' + mergeTaskState.status);
});
```

N/transaction Samples

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/transaction', 'N/config', 'N/record'], function(transaction, config, record) {
    function voidSalesOrder() {
        var accountingConfig = config.load({
            type: config.Type.ACCOUNTING_PREFERENCES
        });
        accountingConfig.setValue({
            fieldId: 'REVERSALVOIDING',
            value: false
        });

        accountingConfig.save();

        var salesOrderObj = record.create({
            type: 'salesorder',
            isDynamic: false
        });
        salesOrderObj.setValue({
            fieldId: 'entity',
            value: 107
        });
        salesOrderObj.setSublistValue({
            sublistId: 'item',
            fieldId: 'item',
            value: 233,
            line: 0
        });
        salesOrderObj.setSublistValue({
            sublistId: 'item',
            fieldId: 'amount',
            value: 1,
            line: 0
        });

        var salesOrderId = salesOrderObj.save();

        var voidSalesOrderId = transaction.void({
            type: record.Type.SALES_ORDER,
            id: salesOrderId
        });

        var salesOrder = record.load({
            type: 'salesorder',
            id: voidSalesOrderId
        });

        // The value of the memo field is 'VOID'
        var memo = salesOrder.getValue({
            fieldId: 'memo'
        });
    }

    voidSalesOrder();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Create a message with translated strings
        var myMsg = message.create({
            title: translation.get({
                collection: 'custcollection_my_strings',
                key: 'MY_TITLE'
            })(),
            message: translation.get({
                collection: 'custcollection_my_strings',
                key: 'HELLO_1'
            })({
                params: ['NetSuite']
            }),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Create a message with translated strings
        var myMsg = message.create({
            title: translation.get({
                collection: 'custcollection_my_strings',
                key: 'MY_TITLE'
            })(),
            message: translation.get({
                collection: 'custcollection_my_strings',
                key: 'MY_MESSAGE'
            })(),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Create a message with translated strings
        var myMsg = message.create({
            title: translation.get({
                collection: 'custcollection_my_strings',
                key: 'MY_TITLE',
                locale: translation.Locale.COMPANY_DEFAULT
            })(),
            message: translation.get({
                collection: 'custcollection_my_strings',
                key: 'MY_MESSAGE',
                locale: translation.Locale.COMPANY_DEFAULT
            })(),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Load translation strings by key
        var localizedStrings = translation.load({
            collections: [{
                alias: 'myCollection',
                collection: 'custcollection_my_strings',
                keys: ['MY_TITLE', 'MY_MESSAGE']
            }]
        });

        // Create a message with translated strings
        var myMsg = message.create({
            title: localizedStrings.myCollection.MY_TITLE(),
            message: localizedStrings.myCollection.MY_MESSAGE(),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

Load Translation Strings by Key from Multiple Translation Collections

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.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Load two Translation Collections
        var localizedStrings = translation.load({
            collections: [{
                alias: 'myCollection',
                collection: 'custcollection_my_strings',
                keys: ['MY_TITLE']
            },{
                alias: 'myOtherCollection',
                collection: 'custcollection_other_strings',
                keys: ['MY_OTHER_MESSAGE']
            }]
        });

        // Create a message with translated strings
        var myMsg = message.create({
            title: localizedStrings.myCollection.MY_TITLE(),
            message: localizedStrings.myOtherCollection.MY_OTHER_MESSAGE(),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

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

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)`.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/ui/message', 'N/translation'],
    function(message, translation) {

        // Load a Translation Collection and a set of locales
        var germanStrings = translation.load({
            collections: [{
                alias: 'myCollection',
                collection: 'custcollection_my_strings',
                keys: ['MY_TITLE', 'MY_MESSAGE']
            }],
            locales: [translation.Locale.de_DE, translation.Locale.es_ES]
        });

        // Select a locale from the list of loaded locales
        var spanishStrings = translation.selectLocale({
            handle: germanStrings,
            locale: translation.Locale.es_ES
        });

        // Create a message with translated strings
        var myMsg = message.create({
            title: germanStrings.myCollection.MY_TITLE(),
            message: spanishStrings.myCollection.MY_MESSAGE(),
            type: message.Type.CONFIRMATION
        });

        // Show the message for 5 seconds
        myMsg.show({
            duration: 5000
        });
});
```

N/ui/dialog Samples

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

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

The following sample shows how to create a confirmation 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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/ui/dialog'], (dialog) => {
    let options = {
        title: 'I am a Confirmation',
        message: 'Press OK or Cancel'
    };

    function success(result) {
        console.log('Success with value ' + result);
    }

    function failure(reason) {
        console.log('Failure: ' + reason);
    }

    dialog.confirm(options).then(success).catch(failure);
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/ui/dialog'], (dialog) => {
    let button1 = {
        label: 'I am A',
        value: 1
    };
    let button2 = {
        label: 'I am B',
        value: 2
    };
    let button3 = {
        label: 'I am C',
        value: 3
    };
    let options = {
        title: 'Alphabet Test',
        message: 'Which One?',
        buttons: [button1, button2, button3]
    };

    function success(result) {
        console.log('Success with value ' + result);
    }
    function failure(reason) {
        console.log('Failure: ' + reason);
    }
    dialog.create(options).then(success).catch(failure);
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/ui/dialog'], (dialog) => {
    let options = {
        title: 'I am a Dialog with the default button',
        message: 'Click a button to continue.',
    };

    function success(result) {
        console.log('Success with value ' + result);
    }

    function failure(reason) {
        console.log('Failure: ' + reason);
    }

    dialog.create(options).then(success).catch(failure);
});
```

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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/ui/dialog'], (dialog) => {
    let options = {
        title: 'I am an Alert',
        message: 'Click OK to continue.'
    };

    function success(result) {
        console.log('Success with value ' + result);
    }

    function failure(reason) {
        console.log('Failure: ' + reason);
    }

    dialog.alert(options).then(success).catch(failure);
});
```

N/ui/message Samples

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

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).

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.

```
/**
 * @NApiVersion 2.1
 */
define(['N/ui/message'], (message) => {
    let myMsg = message.create({
        title: 'My Title',
        message: 'My Message',
        type: message.Type.CONFIRMATION
    });
    myMsg.show({
        duration: 5000 // will disappear after 5s
    });

    let myMsg2 = message.create({
        title: 'My Title 2',
        message: 'My Message 2',
        type: message.Type.INFORMATION
    });
    myMsg2.show();
    setTimeout(myMsg2.hide, 15000); // will disappear after 15s

    let myMsg3 = message.create({
        title: 'My Title 3',
        message: 'My Message 3',
        type: message.Type.WARNING,
        duration: 20000
    });
    myMsg3.show(); // will disappear after 20s

    let myMsg4 = message.create({
        title: 'My Title 4',
        message: 'My Message 4',
        type: message.Type.ERROR
    });
    myMsg4.show(); // will stay up until hide is called.
});
```

N/ui/serverWidget Samples

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

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.

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'], (serverWidget) => {
    const onRequest = (scriptContext) => {
        if (scriptContext.request.method === 'GET') {
            let form = serverWidget.createForm({
                title: 'Simple Form'
            });

            let field = form.addField({
                id: 'textfield',
                type: serverWidget.FieldType.TEXT,
                label: 'Text'
            });
            field.layoutType = serverWidget.FieldLayoutType.NORMAL;
            field.updateBreakType({
                breakType: serverWidget.FieldBreakType.STARTCOL
            });

            form.addField({
                id: 'datefield',
                type: serverWidget.FieldType.DATE,
                label: 'Date'
            });
            form.addField({
                id: 'currencyfield',
                type: serverWidget.FieldType.CURRENCY,
                label: 'Currency'
            });

            let select = form.addField({
                id: 'selectfield',
                type: serverWidget.FieldType.SELECT,
                label: 'Select'
            });
            select.addSelectOption({
                value: 'a',
                text: 'Albert'
            });
            select.addSelectOption({
                value: 'b',
                text: 'Baron'
            });

            let sublist = form.addSublist({
                id: 'sublist',
                type: serverWidget.SublistType.INLINEEDITOR,
                label: 'Inline Editor Sublist'
            });
            sublist.addField({
                id: 'sublist1',
                type: serverWidget.FieldType.DATE,
                label: 'Date'
            });
            sublist.addField({
                id: 'sublist2',
                type: serverWidget.FieldType.TEXT,
                label: 'Text'
            });

            form.addSubmitButton({
                label: 'Submit Button'
            });

            scriptContext.response.writePage(form);
        } else {
            const delimiter = /\u0001/;
            const textField = scriptContext.request.parameters.textfield;
            const dateField = scriptContext.request.parameters.datefield;
            const currencyField = scriptContext.request.parameters.currencyfield;
            const selectField = scriptContext.request.parameters.selectfield;
            const sublistData = scriptContext.request.parameters.sublistdata.split(delimiter);
            const sublistField1 = sublistData[0];
            const sublistField2 = sublistData[1];

            scriptContext.response.write(`You have entered: ${textField} ${dateField} ${currencyField} ${selectField} ${sublist
Field1} ${sublistField2}`);
        }
    }

    return {onRequest}
});
```

Create a Custom Survey Form

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.

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'], function(serverWidget) {
    function onRequest(context) {
        var form = serverWidget.createForm({
            title: 'Thank you for your interest in Wolfe Electronics',
            hideNavBar: true
        });

        var htmlHeader = form.addField({
            id: 'custpage_header',
            type: serverWidget.FieldType.INLINEHTML,
            label: ' '
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
        }).updateBreakType({
            breakType: serverWidget.FieldBreakType.STARTROW
        }).defaultValue = '<p style=\'font-size:20px\'>We pride ourselves on providing the best' +
            ' services and customer satisfaction.  Please take a moment to fill out our survey.</p><br><br>';

        var htmlInstruct = form.addField({
            id: 'custpage_p1',
            type: serverWidget.FieldType.INLINEHTML,
            label: ' '
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
        }).updateBreakType({
            breakType: serverWidget.FieldBreakType.STARTROW
        }).defaultValue = '<p style=\'font-size:14px\'>When answering questions on a scale of 1 to 10,' +
            ' 1 = Greatly Unsatisfied and 10 = Greatly Satisfied.</p><br><br>';

        var productRating = form.addField({
            id: 'custpage_lblproductrating',
            type: serverWidget.FieldType.INLINEHTML,
            label: ' '
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.NORMAL
        }).updateBreakType({
            breakType: serverWidget.FieldBreakType.STARTROW
        }).defaultValue = '<p style=\'font-size:14px\'>How would you rate your satisfaction with our products?</p>';

        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '1',
            source: 'p1'
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.STARTROW
        });
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '2',
            source: 'p2'
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.MIDROW
        });
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '3',
            source: 'p3'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '4',
            source: 'p4'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '5',
            source: 'p5'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '6',
            source: 'p6'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '7',
            source: 'p7'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '8',
            source: 'p8'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '9',
            source: 'p9'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoproductrating',
            type: serverWidget.FieldType.RADIO,
            label: '10',
            source: 'p10'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.ENDROW});

        var serviceRating = form.addField({
            id: 'custpage_lblservicerating',
            type: serverWidget.FieldType.INLINEHTML,
            label: ' '
        }).updateLayoutType({
            layoutType: serverWidget.FieldLayoutType.NORMAL
        }).updateBreakType({
            breakType: serverWidget.FieldBreakType.STARTROW
        }).defaultValue = '<p style=\'font-size:14px\'>How would you rate your satisfaction with our services?</p>';

        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '1',
            source: 'p1'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.STARTROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '2',
            source: 'p2'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '3',
            source: 'p3'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '4',
            source: 'p4'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '5',
            source: 'p5'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '6',
            source: 'p6'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '7',
            source: 'p7'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '8',
            source: 'p8'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '9',
            source: 'p9'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.MIDROW});
        form.addField({
            id: 'custpage_rdoservicerating',
            type: serverWidget.FieldType.RADIO,
            label: '10',
            source: 'p10'
        }).updateLayoutType({layoutType: serverWidget.FieldLayoutType.ENDROW});

        form.addSubmitButton({
            label: 'Submit'
        });

        context.response.writePage(form);
    }

    return {
        onRequest: onRequest
    };
});
```

N/url Samples

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */

// This script creates a URL, sends a secure HTTPS POST request to that URL, and logs the server's response.
require(['N/url', 'N/https'], function(url, https) {
    var script = 'customscript1';
    var deployment = 'customdeploy1';
    var parameters = '';
    try {
        var suiteletURL = url.resolveScript({
            scriptId: script,
            deploymentId: deployment
        });
        var response = https.post({
            url: suiteletURL,
            body: parameters
        });
        log.debug(response.body.toString());
    }
    catch(e) {
        log.error(e.toString());
    }
});
```

Generate an Absolute URL to a Specific Resource

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.

```
/**
 * @NApiVersion 2.x
 */

// This script generates an absolute URL to a specific resource.
require(['N/url', 'N/record'], function(url, record) {
    function resolveRecordUrl() {
        var scheme = 'https://';
        var host = url.resolveDomain({
            hostType: url.HostType.APPLICATION
        });
        var relativePath = url.resolveRecord({
            recordType: record.Type.SALES_ORDER,
            recordId: 6,
            isEditMode: true
        });
        var myURL = scheme + host + relativePath;
    }
    resolveRecordUrl();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

// This script retrieves the domain for calling a RESTlet.
require(['N/url'], function(url) {
    function resolveDomainUrl() {
        var sCompId = 'MSTRWLF';
        var output = url.resolveDomain({
            hostType: url.HostType.RESTLET,
            accountId: sCompId
        });
    }
    resolveDomainUrl();
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

// This script retrieves the relative URL of a record.
require(['N/url'], function(url) {
    var output = url.resolveRecord({
        recordType: 'salesorder',
        recordId: 6,
        isEditMode: true
    });
});
```

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.

```
/**
 * @NApiVersion 2.x
 */
```

```

require(['N/record'], function(record){
    // Create a sales order
    var rec = record.create({
        type: 'salesorder',
        isDynamic: true
    });
    rec.setValue({
        fieldId: 'entity',
        value: 107
    });

    // Set up an object containing an item's internal ID and the corresponding quantity
    var itemList = {
        39: 5,
        38: 1
    }

    // Iterate through the object and set the key-value pairs on the record
    util.each(itemList, function(quantity, itemId){      // (5, 39) and (1, 38)
        rec.selectNewLine('item');
        rec.setCurrentSublistValue('item','item',itemId);
        rec.setCurrentSublistValue('item','quantity',quantity);
        rec.commitLine('item');
    });

    var id = rec.save();
});
```

N/workbook Samples

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

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.

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.

```
/**
 * @NApiVersion 2.x
 */
// The following script creates a simple workbook that contains a simple table
require(['N/workbook', 'N/dataset', 'N/datasetLink'], function(nWorkbook, nDataset, datasetLink) {
    var join = nDataset.createJoin({
        fieldId: "budgetmachine"
    });

    var period = nDataset.createColumn({
        join: join,
        fieldId: "period",
        alias: "budgetmachineperiod",
        label: "Accounting Period"
    });

    var department = nDataset.createColumn({
        fieldId: "department",
        alias: "department",
        label: "Department"
    });

    var total = nDataset.createColumn({
        fieldId: "total",
        alias: "total",
        label: "Amount (Total)"
    });

    var budget = nDataset.create({
        type: 'budgets',
        columns: [period, department, total]
    });

    var postingperiod = nDataset.createColumn({
        fieldId: "postingperiod",
        alias: "postingperiod",
        label: "Posting Period"
    });

    var amount = nDataset.createColumn({
        fieldId: "amount",
        alias: "amount",
        label: "Amount"
    });

    var sales = nDataset.create({
        type: 'salesinvoiced',
        columns: [postingperiod, department, amount],
    });


    var budgetmachineperiod = budget.getExpressionFromColumn({
        alias:"budgetmachineperiod"
    });
    var postingperiodExpression = sales.getExpressionFromColumn({
        alias:"postingperiod"
    });
    var link = datasetLink.create({
        datasets: [budget, sales],
        expressions: [[budgetmachineperiod, postingperiodExpression]],
        id: "link"
    });

    var postingPeriodItem = nWorkbook.createDataDimensionItem({
        expression: postingperiodExpression
    });
    var postingPeriodDimension = nWorkbook.createDataDimension({
        items: [postingPeriodItem]
    });
    var rowSection = nWorkbook.createSection({
        children: [postingPeriodDimension]
    });

    var departmentItem = nWorkbook.createDataDimensionItem({
        expression: budget.getExpressionFromColumn({
            alias: "department"
        })
    });
    var departmentDimension = nWorkbook.createDataDimension({
        items: [departmentItem]
    });

    var sumTotal = nWorkbook.createDataMeasure({
        label: 'Sum Total',
        expression: budget.getExpressionFromColumn({
            alias: 'total'
        }),
        aggregation: 'SUM'
    });

    var sumAmountNet = nWorkbook.createDataMeasure({
        label: 'Sum Amount',
        expression: sales.getExpressionFromColumn({
            alias: 'amount'
        }),
        aggregation: 'SUM'
    });

    var columnSection = nWorkbook.createSection({
        children: [departmentDimension, sumTotal, sumAmountNet]
    });

    var pivot = nWorkbook.createPivot({
        id: "pivot",
        rowAxis:  nWorkbook.createPivotAxis({
            root: rowSection
        }),
        columnAxis: nWorkbook.createPivotAxis({
            root: columnSection
        }),
        name: "Pivot",
        datasetLink: link
    });

    var wb = nWorkbook.create({
        pivots: [pivot]
    });

    wb.runPivot.promise("pivot").then(function(intersections){
        for (var i in intersections)
        {
            var intersection = intersections[i];
            if (intersection.row.itemValues) //skip header
            {
                console.log("Period: " + intersection.row.itemValues[0].value.name);
                console.log(intersection.column.section.children[1].label + ":");
                console.log(intersection.measureValues[0] ? intersection.measureValues[0].value.amount : 0);
                console.log(intersection.column.section.children[2].label + ":");
                console.log(intersection.measureValues[1] ? intersection.measureValues[1].value.amount : 0);
            }
        }
    })
});
```

Create a Comprehensive Workbook

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'.

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.

```
/**
 * @NApiVersion 2.x
 */
```

```
// 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'.

require(['N/workbook', 'N/dataset'], function(workbook, dataset){
    var myDataset = dataset.load({
        id: 'dataset_7'
    });

    var theIDExpression = myDataset.getExpressionFromColumn({
        alias: 'id'
    });
    var sort = workbook.createSort({
        ascending: false
    });
    var columnID = workbook.createTableColumn({
        datasetColumnAlias: 'id'
    });
    var columnName = workbook.createTableColumn({
        datasetColumnAlias: 'name'
    });
    var columnDate = workbook.createTableColumn({
        datasetColumnAlias: 'date'
    });
    var columnTotal = myBasicWorkbook.createTableColumn({
        datasetColumnAlias: 'total'
    });
    var tableView = workbook.createTableDefinition({
        id: 'view',
        name: 'View',
        dataset: myDataset,
        columns: [columnID, columnName, columnDate, columnTotal]
    });
    var theDateExpression = dataset.getExpressionFromColumn({
        alias: 'date'
    });
    var rowItem = workbook.createDataDimensionitem({
        label: 'A',
        expression: theDateExpression
    });
    var rowDataDimension = workbook.createDataDimension({
        items: [rowItem]
    });
    var rowSection = workbook.createSection({
        children: [rowDataDimension]
    });
    var theTotalExpression = dataset.getExpressionFromColumn({
        alias: 'total'
    });
    var columnItem = workbook.createDataDimensionItem({
        label: 'B',
        expression: theTotalExpression
    });    var columnDataDimension = workbook.createDataDimension({
        items: [columnItem]
    });
    var columnMeasure = workbook.createMeasure({
        label: 'M',
        expression: theIDExpression,
        aggregation: workbook.Aggregation.MAX
    });
    var columnSection = workbook.createSection({
        children: [columnDataDimension, columnMeasure]
    });
    var constExpr = workbook.createConstant({
        constant: 1
    });
    var anyOfExpr = workbook.createExpression({
        functionId: workbook.ExpressionType.AND,
        parameters: {
            expression: theIDExpression,
            set: [constExpr]
        }
    });
    var notExpr = workbook.createExpression({
        functionId: workbook.ExpressionType.NOT,
        parameters: {
            a: anyOfExpr
        }
    });
    var allSubNodesSelector = workbook.createAllSubNodesSelector();
    var rowItemSelector = workbook.createDimensionSelector({
        dimension: rowDataDimension
    });
    var columnItemSelector = workbook.createDimensionSelector({
        dimension: columnDataDimension
    });
    var rowSelector = workbook.createPathSelector({
        elements: [allSubNodesSelector, rowItemSelector]
    });
    var columnSelector = workbook.createPathSelector({
        elements: [allSubNodesSelector, columnItemSelector]
    });
    var rowSort = workbook.createDimensionSort({
        item:rowItem,
        sort:sort
    });
    var columnSort = workbook.createMeasureSort({
        measure: columnMeasure,
        sort: sort,
        otherAxisSelector: allSubNodesSelector
    });
    var rowSortDefinition = workbook.createSortDefinition({
        sortBys: [rowSort],
        selector: rowSelector
    });
    var columnSortDefinition = workbook.createSortDefinition({
        sortBys: [columnSort],
        selector: columnSelector
    });
    var rowAxis = workbook.createPivotAxis({
        root: rowSection,
        sortDefinitions: [rowSortDefinition]
    });
    var columnAxis = workbook.createPivotAxis({
        root: columnSection,
        sortDefinitions: [columnSortDefinition]
    });
    var limitingFilter = workbook.createLimitingFilter({
        row: true,
        filteredNodesSelector: rowSelector,
        limit: 1,
        sortBys: [rowSort]
    });
    var conditionalFilter = workbook.createConditionalFilter({
        row: false,
        filteredNodesSelector: rowSelector,
        otherAxisSelector: columnSelector,
        measure: columnMeasure,
        predicate: notExpr
    });
    var pivot = workbook.createPivotDefinition({
        id: 'pivot',
        name: 'Pivot',
        dataset: myDataset,
        rowAxis: rowAxis,
        columnAxis: columnAxis,
        filterExpressions: [notExpr],
        aggregationFilters: [limitingFilter, conditionalFilter]
    });
    var firstAxis = workbook.createChartAxis({
        title: 'First axis'
    });
    var secondAxis = workbook.createChartAxis({
        title: 'Second axis'
    });
    var category = workbook.createCategory({
        axis: firstAxis,
        root: rowSection
    });
    var legend = workbook.createLegend({
        axes: [secondAxis],
        root: columnSection
    });
    var aspect = workbook.createAspect({
        measure: columnMeasure
    });
    var series = workbook.createSeries({
        aspects: [aspect]
    });
    var chart = workbook.createChartDefinition({
        id: 'chart',
        name: 'Chart',
        type: workbook.ChartType.AREA,
        dataset: myDataset,
        category: category,
        legend: legend,
        series: [series]
    });
    var myNewWorkbook = workbook.create({
        description: 'My new updated workbook',
        name: 'Workbook Updated',
        tableDefinitions: [tableView],
        pivotDefinitions: [pivot],
        chartDefinitions: [chart]
    });
    var workbookList = workbook.list();

    log.debug({
        title: "MyNewWorkbook",
        details: myNewWorkbook
    });
});
```

N/workflow Samples

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

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/workflow', 'N/search', 'N/error', 'N/record'],
    function(workflow, search, error, record) {
        function initiateWorkflow() {
            var workflowInstanceId = workflow.initiate({
                recordType: 'customer',
                recordId: 24,
                workflowId: 'customworkflow_myWorkFlow'
            });
            var customerRecord = record.load({
                type: record.Type.CUSTOMER,
                id: 24
            });
        }
        initiateWorkflow();
    });
```

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.

```
/**
```

```
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
require(['N/xml', 'N/file'], function(xml, file) {
    return {
        onRequest: function(options) {
            var sentence = '';
            var xmlFileContent = file.load('SuiteScripts/BookSample.xml').getContents();
            var xmlDocument = xml.Parser.fromString({
                text: xmlFileContent
            });
            var bookNode = xml.XPath.select({
                node: xmlDocument,
                xpath: '//b:book'
            });

            for (var i = 0; i < bookNode.length; i++) {
                var title = bookNode[i].firstChild.nextSibling.textContent;
                var author = bookNode[i].getElementsByTagName({
                    tagName: 'b:author'
                })[0].textContent;
                sentence += 'Author: ' + author + ' wrote ' + title + '.\n';
            }

            options.response.write(sentence);
        }
    };
});
```

This script produces the following output when used with the BookSample.xml file:

```
Author: Giada De Laurentiis wrote Everyday Italian.
Author: J K. Rowling wrote Harry Potter.
Author: James McGovern wrote XQuery Kick Start.
Author: Erik T. Ray wrote Learning XML.
```

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.

```
/**
 * @NApiVersion 2.1
 */

require(['N/crypto/certificate','N/file'],(cert, file) => {
    // Load the file from the File Cabinet.
    // Note that the id value is hard-coded in this sample, and you should use
    // a valid file id from your account.
    let infNFe = file.load({
        id: 922
    });
    let signedXml = cert.signXml({
        algorithm: certificate.HashAlg.SHA256,
        certId: 'custcertificate1',
        rootTag: 'infNFe',
        xmlString: infNFe.getContents()
    });
    cert.verifyXMLSignature({
        signedXml:signedXml,
        rootTag: 'infNFe'
    });
});// Note that this value is hard-coded in this sample, and you should use
```

Modify an XML File

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

The following sample shows how to modify an XML file.

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.

```
/**
 * @NApiVersion 2.x
 */

require(['N/xml','N/file'], function(xml,file) {
    var xmlData = file.load('SuiteScripts/BookSample.xml').getContents();
    var bookShelf = xml.Parser.fromString({
        text: xmlData
        });

    var newBookNode = bookShelf.createElement("book");
    var newTitleNode = bookShelf.createElement("title");
    var newTitleNodeValue = bookShelf.createTextNode("");
    var newAuthorNode = bookShelf.createElement("author");
    var newAuthorNodeValue = bookShelf.createTextNode("");

    newBookNode.appendChild(newTitleNode);
    newBookNode.appendChild(newAuthorNode);
    newTitleNode.appendChild(newTitleNodeValue);
    newAuthorNode.appendChild(newAuthorNodeValue);

});
```

Parse an XML String and Log Element Values

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.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */

require(['N/xml'], function(xml) {
    return {
        onRequest: function(options) {
            var xmlString = '<?xml version="1.0" encoding="UTF-8"?><config date="1465467658668" transient="false">Some content</config>';

            var xmlDocument = xml.Parser.fromString({
                text: xmlString
            });

            var bookNode = xml.XPath.select({
                node: xmlDocument,
                xpath: '//config'
            });

            for (var i = 0; i < bookNode.length; i++) {
                log.debug('Config content', bookNode[i].textContent);
            }
        }
    };
});
```

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.

```
/**
 * @NApiVersion 2.x
 */

// This sample shows how to render an invoice into a PDF file using an XML template in the file cabinet.
// Note that this example requires the Advanced PDF/HTML Templates feature.
require(['N/render', 'N/file', 'N/record'],
    function(render, file, record) {
        function renderRecordToPdfWithTemplate() {
            var xmlTemplateFile = file.load('Templates/PDF Templates/invoicePDFTemplate.xml');
            var renderer = render.create();
            renderer.templateContent = xmlTemplateFile.getContents();
            renderer.addRecord('grecord', record.load({
                type: record.Type.INVOICE,
                id: 37
            }));
            var invoicePdf = renderer.renderAsPdf();
        }
        renderRecordToPdfWithTemplate();
    });
```

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

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

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.

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

define(['N/runtime', 'N/log'], (runtime, log) => {
    function beforeLoad(scriptContext) {
        try {
            const recCurrent = scriptContext.newRecord;
            const objForm = scriptContext.form;
            const stStatus = recCurrent.getValue({
                fieldId: 'status'
            });
            const stSuiteletLinkParam = runtime.getCurrentScript().getParameter({
                name: 'custscript_suiteletlink'
            });
            const suiteletURL = '\"' + stSuiteletLinkParam + '\"';
            if (stStatus === 'Pending Fulfillment') {
                objForm.addButton({
                    id: 'custpage_suiteletbutton',
                    label: 'Open Suitelet',
                    functionName : 'window.open(' + suiteletURL + ')',
                });
            }
        } catch(error) {
            log.error({
                title: 'beforeLoad_addButton',
                details: error.message
            });
        }
    }
    return {
        beforeLoad: beforeLoad
    };
});
```

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.

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

define(['N/record', 'N/log'], (record, log) => {
    function afterSubmit(scriptContext) {
        const stMethodName = 'afterSubmit_calculateCommission';
        try {
            if (scriptContext.type !== scriptContext.UserEventType.CREATE && scriptContext.type !== scriptContext.UserEvent
Type.EDIT) {
                return;
            }
            let stItemType = null;
            let flMSRPTotalAmt = 0.00;
            let flMSRPAmt = 0.00;
            let flNetDistributorCost = 0.00;
            let flCommissionAmount = null;
            let flQuantity = 0.00;
            const recSalesOrder = scriptContext.newRecord;
            let flSubtotal = parseFloat(recSalesOrder.getValue({
                fieldId: 'subtotal'
            }));
            const numItems = recSalesOrder.getLineCount({
                sublistId: 'item'
            });
            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) {
                    flCommissionAmount = flNetDistributorCost * 0.10 + (flMSRPTotalAmt - flNetDistributorCost) * 0.75 + (flSubtotal
 - flMSRPTotalAmt) * 0.5;
                }
            }
            const thisSalesOrderID = recSalesOrder.id;
            const updateSalesOrder = record.load({
                type: record.Type.SALES_ORDER,
                id: thisSalesOrderID
            });
            updateSalesOrder.setValue({
                fieldId: 'custbody_commission_amount',
                value: flCommissionAmount
            });
            const updateSalesOrderID = updateSalesOrder.save();
        } catch(e) {
            log.debug({
                title: stMethodName,
                details: ' - Exit (Catch)- '
            });
            if (e.getDetails !== undefined) {
                log.error({
                    title: 'Process Error',
                    details: JSON.stringify(e)
                });
                throw e;
            } else {
                log.error({
                    title: 'Unexpected Error',
                    details: JSON.stringify(e)
                });
                throw error.create({
                    name: 'Unexpected Error',
                    message: JSON.stringify(e)
                });
            }
        }
    }
    return {
        afterSubmit: afterSubmit
    };
});
```

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 * @NModuleScope SameAccount
 */

define([], function() {
    function fieldChanged(context) {
        try {
            const recInvoice = context.currentRecord;
            const stCurrField = context.fieldId;
            const stCurrSublist = context.sublistId;
            if (stCurrSublist === 'item' && stCurrField === 'custcol_billingitem') {
                let billingitem = recInvoice.getCurrentSublistText({
                    sublistId: 'item',
                    fieldId: 'custcol_billingitem'
                });
                recInvoice.setCurrentSublistText({
                    sublistId: 'item',
                    fieldId: 'item',
                    text: billingitem,
                });
            }
        } catch(e) {
            alert(e.name + ': ' + e.message);
        }
    }
    return {
        fieldChanged: fieldChanged
    };
});
```

Set a Default Posting Period in a Custom Field

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 * @NModuleScope SameAccount
 */

define(['N/runtime', 'N/search', 'N/log'], (runtime, search, log) => {
    function pageInit(context) {
        try {
            if (context.mode === 'copy' || context.mode === 'create') {
                const record = context.currentRecord;
                const searchOpenAccountingPeriods = search.load({
                    id: 'customsearch_open_accounting_periods'
                });
                const resultsFromSearch = searchOpenAccountingPeriods.run().getRange({
                    start: 0,
                    end: 1
                });
                record.setValue({
                    fieldId: 'custbody_preferred_posting_period',
                    value: resultsFromSearch[0].id
                });
                record.setValue({
                    fieldId: 'postingperiod',
                    value: resultsFromSearch[0].id
                });
            }
        } catch(e) {
            log.error({
                title: 'ERROR',
                details: e
            });
        }
    }
    function fieldChanged(context) {
        try {
            if (context.fieldId === 'custbody_preferred_posting_period') {
                const record = context.currentRecord;
                const prefPostPeriod = record.getValue({
                    fieldId: 'custbody_preferred_posting_period'
                });
                record.setValue({
                    fieldId: 'postingperiod',
                    value: prefPostPeriod
                });
            } else {
                return;
            }
        } catch(e) {
            log.error({
                title: 'ERROR',
                details: e
            });
        }
    }
    return {
        pageInit: pageInit,
        fieldChanged: fieldChanged
    };
});
```

Track Deposits and Refunds

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.

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

define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
    function beforeSubmit(scriptContext) {
        const contextDep = scriptContext.newRecord;
        const soID = contextDep.getValue({
            fieldId: 'salesorder'
        });
        if ((soID !== null) && (scriptContext.type === scriptContext.UserEventType.DELETE)) {
            const depAmt = contextDep.getValue({
                fieldId: 'payment'
            });
            const salesorder = record.load({
                type: record.Type.SALES_ORDER,
                id: soID
            });
            const status = salesorder.getValue({
                fieldId: 'status'
            });
            if (status !== 'Billed') {
                const soTotalPaid = salesorder.getValue({
                    fieldId: 'custbody_total_deposit_paid'
                });
                const soRemainingBalance = salesorder.getValue({
                    fieldId: 'custbody_balance_remaining'
                });
                salesorder.setValue({
                    fieldId: 'custbody_total_deposit_paid',
                    value: soTotalPaid - depAmt
                });
                salesorder.setValue({
                    fieldId: 'custbody_balance_remaining',
                    value: (soRemainingBalance + depAmt)
                });
                const id = salesorder.save({
                    enableSourcing: true,
                    ignoreMandatoryFields: true
                });
            }
        }
    }
    function afterSubmit(scriptContext) {
        const contextDep = scriptContext.newRecord;
        const soID = contextDep.getValue({
            fieldId: 'salesorder'
        });
        if ((soID !== null) && ((scriptContext.type === scriptContext.UserEventType.CREATE) || (scriptContext.type === scriptContex
t.UserEventType.EDIT))) {
            const salesorder = record.load({
                type: record.Type.SALES_ORDER,
                id: soID
            });
            const status = salesorder.getValue({
                fieldId: 'status'
            });
            if (status !== 'Billed') {
                const soEntity = salesorder.getValue({
                    fieldId: 'entity'
                });
                const soTranId = salesorder.getValue({
                    fieldId: 'tranid'
                });
                const soFullTextTranID = 'Sales Order #' + soTranId;
                const mySearch = search.load({
                    id: 'customsearch_sobalancedue'
                });
                const entityFilter = search.createFilter({
                    name: 'name',
                    operator: search.Operator.IS,
                    values: soEntity
                });
                const soIdFilter = search.createFilter({
                    name: 'formulatext',
                    operator: search.Operator.IS,
                    summary: search.Summary.MAX,
                    formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application'
 then {createdfrom.salesorder} when {type}='Sales Order' then 'Order #'||{number} end",
                    values: soFullTextTranID
                });
                mySearch.filters.push(entityFilter, soIdFilter);
                const soresults = mySearch.run();
                mySearch.run().each(function(soresults) {
                    let soTextID = soresults.getValue({
                        name: 'formulatext',
                        summary: search.Summary.GROUP
                    });
                    if (soFullTextTranID === soTextID) {
                        let totalPaid = soresults.getValue({
                            name: 'formulacurrency',
                            summary: search.Summary.SUM
                        });
                        let soTotal = salesorder.getValue({
                            fieldId: 'total'
                        });
                        let remainingBalanceOnOrder = parseFloat(soTotal)-parseFloat(totalPaid);
                        salesorder.setValue({
                            fieldId: 'custbody_total_deposit_paid',
                            value: totalPaid
                        });
                        salesorder.setValue({
                            fieldId: 'custbody_balance_remaining',
                            value: remainingBalanceOnOrder
                        });
                        let id = salesorder.save({
                            enableSourcing: true,
                            ignoreMandatoryFields: true
                        });
                    }
                });
            }
        }
    }
    return {
        beforeSubmit: beforeSubmit,
        afterSubmit: afterSubmit
    };
});
```

Important: This sample uses SuiteScript 2.1. For more information, see the help topic SuiteScript 2.1.

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

define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
    function afterSubmit(scriptContext) {
        const contextDepApp = scriptContext.oldRecord;
        const depAppId = contextDepApp.id;
        const soEntity = contextDepApp.getValue({
            fieldId: 'customer'
        });
        const createdFrom = contextDepApp.getValue({
            fieldId: 'deposit'
        });
        const cusDeposit = record.load({
            type: record.Type.CUSTOMER_DEPOSIT,
            id: createdFrom,
            isDynamic: true
        });
        const orderId = cusDeposit.getValue({
            fieldId: 'salesorder'
        });
        const soFullTextTranID = cusDeposit.getText({
            fieldId: 'salesorder',
        });
        const mySearch = search.load({
            id: 'customsearch_sobalancedue'
        });
        const entityFilter = search.createFilter({
            name: 'internalidnumber',
            operator: search.Operator.IS,
            values: soEntity
        });
        const soIdFilter = search.createFilter({
            name: 'formulatext',
            operator: search.Operator.IS,
            formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then {creat
edfrom.salesorder} when {type}='Sales Order' then 'Sales Order #'||{number} end",
            values: soFullTextTranID
        });
        mySearch.filters.push(entityFilter, soIdFilter);
        const soresults = mySearch.run();
        mySearch.run().each(function(soresults) {
            let soTextID = soresults.getValue({
                name: 'formulatext',
                summary: search.Summary.GROUP
            });
            if (soFullTextTranID === soTextID) {
                let totalPaid = soresults.getValue({
                    name: 'formulacurrency',
                    summary: search.Summary.SUM
                });
                let salesorder = record.load({
                    type: record.Type.SALES_ORDER,
                    id: orderId,
                    isDynamic: true
                });
                let soTotal = salesorder.getValue({
                    fieldId: 'total'
                });
                let remainingBalanceOnOrder = parseFloat(soTotal);
                remainingBalanceOnOrder = parseFloat(remainingBalanceOnOrder) - parseFloat(totalPaid);
                salesorder.setValue({
                    fieldId: 'custbody_total_deposit_paid',
                    value: totalPaid
                });
                salesorder.setValue({
                    fieldId: 'custbody_balance_remaining',
                    value: remainingBalanceOnOrder
                });
                let id = salesorder.save({
                    enableSourcing: true,
                    ignoreMandatoryFields: true
                });
            }
        });
    }
    return {
        afterSubmit: afterSubmit
    };
});
```

Important: This sample uses SuiteScript 2.1. For more information, see the help topic SuiteScript 2.1.

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

define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
    return {
        afterSubmit: function(scriptContext) {
            const contextRef = scriptContext.newRecord;
            const refId = contextRef.id;
            UpdateSalesOrder(refId);
        }
    };
    function UpdateSalesOrder(refId) {
        const refund = record.load({
            type: record.Type.CUSTOMER_REFUND,
            id: refId,
            isDynamic: true
        });
        const soEntity = refund.getValue({
            fieldId: 'customer'
        });
        const lines = refund.getLineCount({
            sublistId: 'apply'
        });
        for (let i = 0; i < lines; i++) {
            let depositnum = refund.getSublistText({
                sublistId: 'apply',
                fieldId: 'internalid',
                line: i
            });
            let refundamt = refund.getSublistValue({
                sublistId: 'apply',
                fieldId: 'amount',
                line: i
            });
            let order = search.lookupFields({
                type: search.Type.DEPOSIT_APPLICATION,
                id: depositnum,
                columns: 'createdfrom.salesorder'
            });
            let soFullTextTranID = order['createdfrom.salesorder'][0].text;
            let orderId = order['createdfrom.salesorder'][0].value;
            let soTotalPaid = search.lookupFields({
                type: search.Type.SALES_ORDER,
                id: orderId,
                columns: ['total']
            });
            let soTotal = soTotalPaid['total'];
            let mySearch = search.load({
                id: 'customsearch_sobalancedue'
            });
            let entityFilter = search.createFilter({
                name: 'internalid',
                join: 'customer',
                operator: search.Operator.EQUALTO,
                summary: search.Summary.MAX,
                values: soEntity
            });
            let soIdFilter = search.createFilter({
                name: 'formulatext',
                operator: search.Operator.IS,
                formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then
 {createdfrom.salesorder} when {type}='Sales Order' then 'Sales Order #'||{number} end",
                values: soFullTextTranID
            });
            mySearch.filters.push(entityFilter, soIdFilter);
            let soresults = mySearch.run();
            mySearch.run().each(function(soresults) {
                let soTextID = soresults.getValue({
                    name: 'formulatext',
                    summary: search.Summary.GROUP
                });
                if (soFullTextTranID === soTextID) {
                    let totalPaid = soresults.getValue({
                        name: 'formulacurrency',
                        summary: search.Summary.SUM
                    });
                    let remainingBalanceOnOrder = parseFloat(soTotal);
                    remainingBalanceOnOrder = parseFloat(remainingBalanceOnOrder)-parseFloat(totalPaid);
                    let salesorder = record.load({
                        type: record.Type.SALES_ORDER,
                        id: orderId,
                        isDynamic: true
                    });
                    salesorder.setValue({
                        fieldId: 'custbody_total_deposit_paid',
                        value: totalPaid
                    });
                    salesorder.setValue({
                        fieldId: 'custbody_balance_remaining',
                        value: remainingBalanceOnOrder
                    });
                    let id = salesorder.save({
                        enableSourcing: true,
                        ignoreMandatoryFields: true
                    });
                }
            });
        }
    }
});
```

Important: This sample uses SuiteScript 2.1. For more information, see the help topic SuiteScript 2.1.

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

define(['N/record', 'N/search', 'N/log'], (record, search, log) => {
    function afterSubmit(scriptContext) {
        const contextOrder = scriptContext.newRecord;
        const soID = contextOrder.id;
        const salesorder = record.load({
            type: record.Type.SALES_ORDER,
            id: soID
        });
        const soTotal = salesorder.getValue({
            fieldId: 'total'
        });
        const soEntity = salesorder.getValue({
            fieldId: 'entity'
        });
        const soTranId = salesorder.getValue({
            fieldId: 'tranid'
        });
        const soFullTextTranID = 'Sales Order #'+soTranId;
        const mySearch = search.load({
            id: 'customsearch_sobalancedue'
        });
        const entityFilter = search.createFilter({
            name: 'entity',
            operator: search.Operator.ANYOF,
            values: soEntity
        });
        const soIdFilter = search.createFilter({
            name: 'formulatext',
            operator: search.Operator.IS,
            formula: "CASE WHEN {type}='Customer Deposit' then {appliedtotransaction} when {type}='Deposit Application' then {createdfrom.salesorder} when {type}='Sales Order' then 'Sales Order #'||{number} end",
            values: soFullTextTranID
        });
        mySearch.filters.push(entityFilter, soIdFilter);
        const soresults = mySearch.run();
        mySearch.run().each(function(soresults) {
            let soTextID = soresults.getValue({
                name: 'formulatext',
                summary: search.Summary.GROUP
            });
            if (soFullTextTranID === soTextID) {
                let totalPaid = soresults.getValue({
                    name: 'formulacurrency',
                    summary: search.Summary.SUM
                });
                let remainingBalanceOnOrder = parseFloat(parseFloat(soTotal))-parseFloat(totalPaid);
                salesorder.setValue({
                    fieldId: 'custbody_total_deposit_paid',
                    value: totalPaid
                });
                salesorder.setValue({
                    fieldId: 'custbody_balance_remaining',
                    value: remainingBalanceOnOrder
                });
                let id = salesorder.save({
                    enableSourcing: true,
                    ignoreMandatoryFields: true
                });
            }
        });
    }
    return{
        afterSubmit: afterSubmit
    };
});
```

Set the Purchase Order Exchange Rate

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.

```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 * @NModuleScope SameAccount
 */

define(['N/runtime', 'N/currentRecord', 'N/currency', 'N/log'], (runtime, currentRecord, currency, log) => {
    function saveRecord(context) {
        try {
            const stUserCurrency = runtime.getCurrentScript().getParameter({
                name: 'custscript_custom_currency'
            });
            if (stUserCurrency === " " || stUserCurrency === null || stUserCurrency === undefined) {
                throw "Please enter a value for Custom Currency at Home > User Preferences > Custom.";
            }
            const purchaseOrder = context.currentRecord;
            const stTranCurrency = purchaseOrder.getValue({
                fieldId: 'currency'
            });
            const stTranDate = purchaseOrder.getValue({
                fieldId: 'trandate'
            });
            const stTotal = purchaseOrder.getValue({
                fieldId: 'total'
            });
            let flTotalAmount = parseFloat(stTotal);
            let exchangeRate = currency.exchangeRate({
                source: stTranCurrency,
                target: stUserCurrency,
                date: stTranDate
            });
            const flExchangeRate = parseFloat(exchangeRate);
            const flAmountInUserCurrency = parseFloat(flTotalAmount * flExchangeRate);
            purchaseOrder.setValue({
                fieldId: 'custbody_currency_exchange_rate',
                value: flExchangeRate
            });
            purchaseOrder.setValue({
                fieldId: 'custbody_currency_po_amount',
                value: flAmountInUserCurrency
            });
        } catch(e) {
            if (e.getDetails !== undefined) {
                log.error({
                    title: 'Process Error',
                    details: JSON.stringify(e)
                });
            } else {
                log.error({
                    title: 'Unexpected Error',
                    details: JSON.stringify(e)
                });
            }
            throw (e);
        }
        return true;
    }
    return {
        saveRecord: saveRecord
    };
});
```

Custom Plug-in Samples

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

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 Adminstrator 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 SuiteScript Debugger.

The following code is a sample validation plug-in script.

```
/**
 * Copyright (c) 2017, Oracle and/or its affiliates.
 *
 * @NModuleScope public
 * @NApiVersion 2.x
 * @NScriptType plugintypeimpl
 */

define([], function() {
    /**
   * validate - This function is the entry point of our plugin script
```

```
     * @param {Object} plugInContext
     * @param {Object} plugInContext.eDocument
     * @param {String} plugInContext.eDocument.id
     * @param {String} plugInContext.eDocument.scriptId
     * @param {String} plugInContext.eDocument.content
     * @param {Object} plugInContext.eDocument.source
     * @param {String} plugInContext.eDocument.source.id
     * @param {String} plugInContext.eDocument.source.text
     * @param {Object} plugInContext.eDocument.template
     * @param {String} plugInContext.eDocument.template.id
     * @param {String} plugInContext.eDocument.template.text
     * @param {Object} plugInContext.eDocument.status
     * @param {Integer} plugInContext.eDocument.status.id
     * @param {String} plugInContext.eDocument.status.text
     * @param {Object} plugInContext.eDocument.package
     * @param {String} plugInContext.eDocument.package.id
     * @param {String} plugInContext.eDocument.package.text
     * @param {Object} plugInContext.eDocument.transactionType
     * @param {String} plugInContext.eDocument.transactionType.id
     * @param {String} plugInContext.eDocument.transactionType.text
     * @param {Object} plugInContext.eDocument.vendor
     * @param {String} plugInContext.eDocument.vendor.id
     * @param {String} plugInContext.eDocument.vendor.text
     * @returns {Object} result
     * @returns {Boolean} result.success
     * @returns {String} result.message
     */
    function validate(pluginContext) {

        var eDoc = pluginContext.eDocument;
        var result = {
                success: false,
                message: ""
        };


        // Connect to validation service


        // If successful
        result.success = true;
        result.message = "Validation successful!";


        // Sample result if not successful
        // result.success = false;
        // result.message = "Service returned a failed response";


        return result;
    }


    return {
        validate: validate
    };

});
```

Create a Custom Plug-in Implementation for E-Document Custom Data Source

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.

```
* @NApiVersion 2.x
 * @NScriptType plugintypeimpl
 * @NModuleScope Public
 */
define(["N/render"], function(nsrender) {
  /**
  * inject - This function will provide the custom data source during the generation process
  * @param {Object} params
  * @param {String} params.transactionId
  * @param {Object} params.transactionRecord
  * @param {Number} params.userId
  *
  * @returns {Object} result
  * @returns {render.DataSource} result.alias
  * @returns {string} result.format
  * @returns {Object | Document | string} result.data
  */

  function inject(params) {
    var txnRecord = params.transactionRecord;
    var txnId = params.transactionId;
    var userId = params.userId
    var customObj = {};
    log.debug("Custom Object", customObj);
    return {
      customDataSources: [
        {
          format: nsrender.DataSource.OBJECT,
          alias: "custom",
          data: customObj
        }
      ],
    };
  }

  return {
    inject: inject
  };
});
```

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 Adminstrator 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.

```
/**
 * @NApiVersion 2.0
 * @NScriptType plugintypeimpl
 */
define(["../../lib/string_formatter",
   "../../lib/wrapper/ns_wrapper_error",
   "../../lib/wrapper/ns_wrapper_config",
   "../../lib/wrapper/ns_wrapper_email",
   "../../lib/wrapper/ns_wrapper_file",
   "../../app/einvoice/app_einvoice_notifier",
   "../../lib/translator"],
function(stringFormatter, error, config, email, file, notifier, translator) {

         /**
      * send - This function is the entry point of our plugin script
      * @param {Object} plugInContext
      * @param {String} plugInContext.scriptId
      * @param {String} plugInContext.sendMethodId
      * @param {String} plugInContext.eInvoiceContent
      * @param {Object} plugInContext.customer
      * @param {String} plugInContext.customer.id
      * @param {Array}  plugInContext.customer.recipients
      * @param {Object} plugInContext.transaction
      * @param {String} plugInContext.transaction.number
      * @param {String} plugInContext.transaction.id
      * @param {String} plugInContext.transaction.poNum
      * @param {Object} plugInContext.sender
      * @param {String} plugInContext.sender.id
      * @param {String} plugInContext.sender.name
      * @param {String} plugInContext.sender.email
      * @param {Array} plugInContext.attachmentFileIds
      *
      * @returns {Object} result
      * @returns {Boolean} result.success
      * @returns {String} result.message
      */
        function send(pluginContext) {

            var MSG_NO_EMAIL = translator.getString("ei.sending.sendernoemail");
            var MSG_SENT_DETAILS = translator.getString("ei.sending.sentdetails");

            var senderDetails = pluginContext.sender;
            var customer = pluginContext.customer;
            var transaction = pluginContext.transaction;
            var recipientList = customer.recipients;
            var result = {};
            var parameters;
            if (!senderDetails.email) {
                parameters = {
                    EMPLOYEENAME: senderDetails.name
                };
                stringFormatter.setString(MSG_NO_EMAIL);
                stringFormatter.replaceParameters(parameters);
                result = {
                    success: false,
                    message: stringFormatter.toString()
                };
            } else {
                var invoiceSendDetails = {
                    number: transaction.number,
                    poNumber: transaction.poNum,
                    transactionType : transaction.type,
                    eInvoiceContent: pluginContext.eInvoiceContent,
                    attachmentFileIds: pluginContext.attachmentFileIds
                };
                notifier.notifyRecipient(senderDetails.id, recipientList, invoiceSendDetails);

                parameters = {
                    SENDER: senderDetails.email,
                    RECIPIENTS: recipientList.join(", ")
                };
                stringFormatter.setString(MSG_SENT_DETAILS);
                stringFormatter.replaceParameters(parameters);

                result = {
                    success: true,
                    message: stringFormatter.toString()
                };

            }

            return result;

        }

        return {
            send: send
        };
    });
```

Create a Digital Signature Plug-in Implementation for E-Documents

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 Adminstrator 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 digital signature plug-in implementation.

```
/**
 * Copyright (c) 2019, Oracle NetSuite and/or its affiliates.
 *
 * @NApiVersion 2.x
 * @NModuleScope Public
 * @NScriptType plugintypeimpl
 */
define(['N/crypto/certificate','N/file'],

    function(certificate, file){

        /**
       *
       * @param pluginContext
       * @param {String} pluginContext.unsignedString
       * @param {String} plugincontext.subsidiaryId
       *
       * @returns {Object} result
       * @returns {string} result.success
       * @returns {String} result.signedString
       * @returns {String} result.message
```

```
  */
   function signDocument(pluginContext){

       /**
     * Extract the values from pluginContext
     */
       var unsignedString = pluginContext.unsignedString;
       var subsidiaryId = pluginContext.subsidiaryId;

       var rootTag = "RootTag";
       var certificateId = "custcertificatesfd";
       var algorithm = "SHA1";

       var result = { success : true, signedString : unsignedString, message : "This is default implementation of Digital Signa
ture."};

       /**
     * Call services to sign the string
     */
       try {

           var random = 0;
           /*var signedXML = certificate.signXml({
               algorithm : algorithm,
               certId: certificateId,
               rootTag : rootTag,
               xmlString : unsignedString
           });

           result.success  = true;
           result.signedString = signedXML.asString();
           result.message = "Document signed successfully";
*/
       }catch(e){
           result.success = false;
           result.signedString = "";
           result.message = e.message;
       }


       return result;
   }

   return {
       signDocument : signDocument
   };
});
```

Create a Quality Custom Inspection Rule Plug-in

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.

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.

```
/**
* @NApiVersion 2.x
* @NScriptType plugintypeimpl
*
* ECHO Implementation of Custom Rule Plug-in for Quality Management SuiteApp
  */
define(['N/log'], function(log) {
      return {
         inspectionPassed: function(inspectionObj, fieldObj, otherFieldObjs, standardObjs) {
            // echo inspectionObj
            log.debug({
                     title: 'insepctionObj',
                     details: 'type:' + inspectionObj.type +
                           ' name:' + inspectionObj.name +
                           ' txnId:' + inspectionObj.txnId +
                           ' itemId:' + inspectionObj.itemId
            });

            // echo fieldObj
            log.debug({
                     title: 'fieldObj',
                     details: 'name:' + fieldObj.name +
                              ' value:' + fieldObj.value
            });

            // echo otherFieldObjs
            for ( var i in otherFieldObjs ) {
                     log.debug({
                                    title: 'otherFieldObjs',
                                    details: 'name:' + otherFieldObjs[i].name +
                                                   ' value:' + otherFieldObjs[i].value
                     });
            }

            // echo standardObjs
            for ( var i in standardObjs ) {
                  log.debug({
                              title: 'standardObjs',
                              details: 'name:' + standardObjs[i].name +
                                             ' value:' + standardObjs[i].value
                  });
            }
            return true;
         }
      }
});
```

Create a Script for Sending E-Documents

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 Adminstrator 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 for sending e-documents.

```
/**
```

```
 * @NApiVersion 2.x
 * @NModuleScope Public
 */
define(["N/record"], function(record, error) {
    return {
        /**
    * send - Sample implementation: This will copy the e-document content to the document's
    Memo field
    *
    * @param {Object} plugInContext
    * @param {String} plugInContext.scriptId
    * @param {String} plugInContext.sendMethodId
    * @param {String} plugInContext.eInvoiceContent
    *
    * @param {Object} plugInContext.customer
    * @param {String} plugInContext.customer.id
    * @param {String[]} plugInContext.customer.recipients
    *
    * @param {Object} plugInContext.transaction
    * @param {String} plugInContext.transaction.id
    * @param {String} plugInContext.transaction.number
    * @param {String} plugInContext.transaction.poNum
    *
    * @param {Object} plugInContext.sender
    * @param {String} plugInContext.sender.id
    * @param {String} plugInContext.sender.name
    * @param {String} plugInContext.sender.email
    *
    *
    * @returns {Object} result
    * @returns {Boolean} result.success: determines
    * @returns {String} result.message: a failure message
    */
        send: function(plugInContext) {
            var result = {
                success: true,
                message: "Success"
            };
            try {
                var rec = record.load({
                    type: record.Type.INVOICE,
                    id: plugInContext.transaction.id,
                });
                rec.setValue({
                    fieldId: "memo",
                    value: [
                        "Script ID: " + plugInContext.scriptId,
                        "Customer: " + plugInContext.customer.name,
                        "Transaction: " + plugInContext.transaction.number,
                        "Sender: " + plugInContext.sender.name,
                        "Recipients: " + plugInContext.customer.recipients.join("\n"),
                        "Content: " + plugInContext.eInvoiceContent].join("\n\n")
                });
                rec.save();
            } catch (e) {
                result.success = false;
                result.message = "Failure";
            }
            return result;
        }
    };
});
```

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.

```
/**
* Copyright (c) 2017, Oracle and/or its affiliates.
*
* @NApiVersion 2.x
* @NScriptType plugintypeimpl
* @NModuleScope public
*/
define([], function() {
    /**
    * send - This function is the entry point of our plugin script
    * @param {Object} plugInContext
    * @param {String} plugInContext.scriptId
    * @param {String} plugInContext.sendMethodId
    * @param {String} plugInContext.eInvoiceContent
    * @param {Object} plugInContext.customer
    * @param {String} plugInContext.customer.id
    * @param {Array} plugInContext.customer.recipients
    * @param {Object} plugInContext.transaction
    * @param {String} plugInContext.transaction.number
    * @param {String} plugInContext.transaction.id
    * @param {String} plugInContext.transaction.poNum
    * @param {Object} plugInContext.sender
    * @param {String} plugInContext.sender.id
    * @param {String} plugInContext.sender.name
    * @param {String} plugInContext.sender.email
    * @param {Array} plugInContext.attachmentFileIds
    *
    * @returns {Object} result
    * @returns {Boolean} result.success
    * @returns {String} result.message
    */
    return {
        send: function(pluginContext) {
            var result = {
                success: true,
                message: '',
                eiStatus: {
                    "transactionId": plugInContext.transaction.id,
                    "transactionType": plugInContext.transaction.tranType,
                    "entity": customer.id,
                    "eDocStatus": "3",
                    "eventType": "3",
                    "details": "The e-Doc successfully certified and is ready for sending.",
                    "owner": plugInContext.sender.id,
                    "isUpdateFields": "true",
                    "extraFieldsForUpdate": {},
                    "bundleId": "",
                    "bundleName": ""
                }
            };
            return result;
        }
    }
});
```