

SuiteScript 2.0 API



Copyright © 2005, 2018, Oracle and/or its affiliates. All rights reserved.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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

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

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

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

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

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

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

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

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

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

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

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

Sample Code

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

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

No Excessive Use of the Service

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

Beta Features

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

Table of Contents

SuiteScript 2.0 API Introduction	1
SuiteScript 2.0 Hello World	1
SuiteScript 2.0 Script Basics	10
SuiteScript 2.0 Anatomy of a Script	12
SuiteScript 2.0 Script Creation Process	14
SuiteScript 2.0 Advantages	15
SuiteScript 2.0 Terminology	18
SuiteScript 2.0 Developer Resources	19
SuiteScript Versioning Guidelines	20
SuiteScript 1.0 to SuiteScript 2.0 API Map	22
SuiteScript 1.0 to SuiteScript 2.0 API Map – Functions (nlapi)	22
SuiteScript 1.0 to SuiteScript 2.0 API Map – Objects (nlobj)	34
SuiteScript 2.0 Script Types	60
SuiteScript 2.0 Bundle Installation Script Type	62
SuiteScript 2.0 Bundle Installation Script Entry Points	62
SuiteScript 2.0 Client Script Type	64
SuiteScript 2.0 Client Script Reference	69
SuiteScript 2.0 Client Script Entry Points and API	73
SuiteScript 2.0 Map/Reduce Script Type	79
SuiteScript 2.0 Map/Reduce Script Reference	90
SuiteScript 2.0 Map/Reduce Script Entry Points and API	123
SuiteScript 2.0 Mass Update Script Type	160
SuiteScript 2.0 Mass Update Script Entry Points	161
SuiteScript 2.0 Portlet Script Type	162
SuiteScript 2.0 Portlet Script Reference	166
SuiteScript 2.0 Portlet Script Entry Points and API	168
SuiteScript 2.0 RESTlet Script Type	177
SuiteScript 2.0 Getting Started with RESTlets	177
SuiteScript 2.0 RESTlet Reference	183
SuiteScript 2.0 RESTlet Script and Request Samples	185
SuiteScript 2.0 RESTlet Script Entry Points	192
SuiteScript 2.0 Scheduled Script Type	194
SuiteScript 2.0 Scheduled Script Reference	197
SuiteScript 2.0 Script Entry Points and API	207
SuiteScript 2.0 Suitelet Script Type	208
SuiteScript 2.0 Suitelet Script Reference	210
SuiteScript 2.0 Suitelet Script Entry Points and API	219
SuiteScript 2.0 User Event Script Type	220
SuiteScript 2.0 User Event Script Reference	222
SuiteScript 2.0 User Event Script Entry Points and API	233
SuiteScript 2.0 Workflow Action Script Type	237
SuiteScript 2.0 Workflow Action Script Entry Points and API	239
SuiteScript 2.0 Global Objects	240
define Object	240
define(moduleObject)	241
define(id, [dependencies,] moduleObject)	242
require Function	245
require([dependencies,] callback)	245
require Configuration	247
log Object	248
util Object	248
toString()	249
JSON object	249

JSON.parse(text)	250
JSON.stringify(obj)	250
Promise Object	251
SuiteScript 2.0 Modules	255
N/action Module	257
action.Action	259
action.get(options)	266
action.get.promise(options)	267
action.find(options)	268
action.find.promise(options)	270
action.execute(options)	271
action.execute.promise(options)	272
N/auth Module	273
auth.changeEmail(options)	274
auth.changePassword(options)	275
N/cache Module	276
cache.Cache	279
cache.getCache(options)	284
cache.Scope	285
N/config Module	286
config.load(options)	287
config.Type	288
N/crypto Module	290
crypto.Cipher	294
crypto.CipherPayload	296
crypto.Decipher	298
crypto.Hash	300
crypto.Hmac	302
crypto.SecretKey	304
crypto.createCipher(options)	305
crypto.createDecipher(options)	306
crypto.createHash(options)	307
crypto.createHmac(options)	308
crypto.createSecretKey(options)	309
crypto.EncryptionAlg	310
crypto.HashAlg	310
crypto.Padding	311
N/currency Module	312
currency.exchangeRate(options)	313
N/currentRecord Module	314
currentRecord.Column	322
currentRecord.CurrentRecord	324
currentRecord.Field	365
currentRecord.Sublist	374
currentRecord.get()	377
currentRecord.get.promise()	378
N/email Module	379
email.send(options)	380
email.send.promise(options)	384
email.sendBulk(options)	384
email.sendBulk.promise(options)	387
email.sendCampaignEvent(options)	388
email.sendCampaignEvent.promise(options)	389
N/encode Module	389
encode.convert(options)	390

encode.Encoding	391
N/error Module	392
error.SuiteScriptError	394
error.UserEventError	397
error.create(options)	401
N/file Module	402
file.File	406
file.create(options)	420
file.delete(options)	422
file.load(options)	423
file.Encoding	424
file.Type	425
N/format Module	426
format.format(options)	428
format.parse(options)	430
format.Type	431
format.Timezone	432
N/http Module	436
http.ClientResponse	440
http.ServerRequest	443
http.ServerResponse	450
http.get(options)	461
http.get.promise(options)	462
http.delete(options)	462
http.delete.promise(options)	463
http.request(options)	464
http.request.promise(options)	465
http.post(options)	466
http.post.promise(options)	467
http.put(options)	468
http.put.promise(options)	469
http.CacheDuration	470
http.Method	471
http.RedirectType	472
N/https Module	472
https.SecureString	479
https.createSecureKey(options)	483
https.createSecureKey.promise(options)	484
https.createSecureString(options)	485
https.createSecureString.promise(options)	485
https.ClientResponse	486
https.ServerRequest	489
https.ServerResponse	495
https.get(options)	506
https.get.promise(options)	507
https.delete(options)	507
https.delete.promise(options)	508
https.request(options)	509
https.request.promise(options)	510
https.post(options)	511
https.post.promise(options)	512
https.put(options)	513
https.put.promise(options)	514
https.CacheDuration	515
https.Encoding	516

https.Method	516
https.RedirectType	517
N/log Module	518
log.audit(options)	520
log.debug(options)	521
log.emergency(options)	522
log.error(options)	523
N/plugin Module	524
plugin.findImplementations(options)	526
plugin.loadImplementation(options)	526
N/portlet Module	527
portlet.resize	529
portlet.refresh	530
N/query Module	530
Scripting with the N/query Module	543
query.Column	551
query.Component	554
query.Condition	570
query.Page	574
query.PagedData	576
query.PageRange	578
query.Query	580
query.Result	606
query.ResultSet	607
query.Sort	609
query.create(options)	614
query.delete(options)	616
query.load(options)	617
query.Aggregate	618
query.Operator	619
query.ReturnType	621
query.SortLocale	622
query.Type	628
N/record Module	646
record.Column	667
record.Field	671
record.Macro	677
record.Record	683
record.Sublist	741
record.attach(options)	745
record.attach.promise(options)	747
record.copy(options)	749
record.copy.promise(options)	750
record.create(options)	752
record.create.promise(options)	754
record.delete(options)	756
record.delete.promise(options)	757
record.detach(options)	759
record.detach.promise(options)	760
record.load(options)	762
record.load.promise(options)	764
record.submitFields(options)	766
record.submitFields.promise(options)	768
record.transform(options)	770
record.transform.promise(options)	773

record.Type	774
N/redirect Module	777
redirect.redirect(options)	779
redirect.toRecord(options)	780
redirect.toSavedSearch(options)	781
redirect.toSavedSearchResult(options)	781
redirect.toSearch(options)	782
redirect.toSearchResult(options)	783
redirect.toSuitelet(options)	783
redirect.toTaskLink(options)	784
N/render Module	785
render.EmailMergeResult	789
render.TemplateRenderer	791
render.bom(options)	799
render.create()	800
render.mergeEmail(options)	800
render.packingSlip(options)	802
render.pickingTicket(options)	802
render.statement(options)	803
render.transaction(options)	804
render.xmlToPdf(options)	805
render.DataSource	806
render.PrintMode	807
N/runtime Module	808
runtime.Script	812
runtime.Session	817
runtime.User	819
runtime.getCurrentScript()	825
runtime.getCurrentSession()	826
runtime.getCurrentUser()	826
runtime.isFeatureInEffect(options)	827
runtime.accountId	828
runtime.envType	828
runtime.executionContext	829
runtime.processorCount	829
runtime.queueCount	830
runtime.version	831
runtime.ContextType	831
runtime.EnvType	832
runtime.Permission	833
N/search Module	833
search.Search	844
search.Result	855
search.Column	862
search.Filter	869
search.ResultSet	872
search.Page	877
search.PagedData	883
search.PageRange	887
search.Setting	889
search.create(options)	891
search.create.promise(options)	895
search.createSetting(options)	895
search.load(options)	897
search.load.promise(options)	899

search.delete(options)	900
search.delete.promise(options)	901
search.duplicates(options)	902
search.duplicates.promise(options)	903
search.global(options)	904
search.global.promise(options)	905
search.lookupFields(options)	906
search.lookupFields.promise(options)	907
search.createColumn(options)	908
search.createFilter(options)	910
search.Operator	911
search.Sort	912
search.Summary	913
search.Type	913
N/sftp Module	916
Setting up an SFTP Transfer	919
SFTP Authentication	920
Supported Cipher Suites and Host Key Types	922
Supported SuiteScript File Types	922
sftp.Connection	924
sftp.createConnection(options)	928
N/sso Module	930
sso.generateSuiteSignOnToken(options)	931
N/task Module	933
task.ScheduledScriptTask	948
task.ScheduledScriptTaskStatus	951
task.MapReduceScriptTask	953
task.MapReduceScriptTaskStatus	956
task.CsvImportTask	967
task.CsvImportTaskStatus	972
task.EntityDeduplicationTask	973
task.EntityDeduplicationTaskStatus	977
task.SearchTask	979
task.SearchTaskStatus	986
task.WorkflowTriggerTask	989
task.WorkflowTriggerTaskStatus	992
task.create(options)	993
task.checkStatus(options)	998
task.TaskType	998
task.TaskStatus	999
task.MasterSelectionMode	1000
task.DedupeMode	1001
task.DedupeEntityType	1002
task.MapReduceStage	1002
N/transaction Module	1003
transaction.void(options)	1005
transaction.void.promise(options)	1006
transaction.Type	1007
N/ui/dialog Module	1008
dialog.alert(options)	1010
dialog.confirm(options)	1011
dialog.create(options)	1012
N/ui/message module	1013
message.Message	1015
message.create(options)	1016

message.Type	1017
N/ui/serverWidget Module	1018
serverWidget.Assistant	1032
serverWidget.AssistantStep	1055
serverWidget.Button	1064
serverWidget.Field	1066
serverWidget.FieldGroup	1081
serverWidget.Form	1086
serverWidget.List	1112
serverWidget.ListColumn	1121
serverWidget.Sublist	1124
serverWidget.Tab	1135
serverWidget.createAssistant(options)	1137
serverWidget.createForm(options)	1138
serverWidget.createList(options)	1139
serverWidget.AssistantSubmitAction	1140
serverWidget.FieldBreakType	1141
serverWidget.FieldDisplayType	1142
serverWidget.FieldLayoutType	1143
serverWidget.FieldType	1144
serverWidget.FormPageLinkType	1145
serverWidget.LayoutJustification	1146
serverWidget.ListStyle	1147
serverWidget.SublistDisplayType	1147
serverWidget.SublistType	1148
N/url Module	1150
url.format(options)	1152
url.resolveDomain(options)	1153
url.resolveRecord(options)	1154
url.resolveScript(options)	1155
url.resolveTaskLink(options)	1156
url.HostType	1156
N/util Module	1157
util.isArray(obj)	1159
util.isBoolean(obj)	1160
util.isDate(obj)	1161
util.isFunction(obj)	1162
util.isNumber(obj)	1162
util.isObject(obj)	1163
util.isRegExp(obj)	1164
util.isString(obj)	1164
util.nanoTime()	1165
util.each(iterable, callback)	1166
util.extend(receiver, contributor)	1166
N/workflow Module	1168
workflow.initiate(options)	1169
workflow.trigger(options)	1170
N/xml Module	1171
xml.Parser	1180
xml.XPath	1182
xml.Node	1183
xml.Document	1200
xml.Element	1215
xml.Attr	1227
xml.escape(options)	1230

xml.validate(options)	1230
xml.NodeType	1231
SuiteScript 2.0 Record Actions and Macros	1233
Overview of Record Action and Macro APIs	1233
Supported Record Actions	1233
Supported Record Macros	1234
SuiteScript 2.0 JSDoc Validation	1235
Controlling Access to Scripts and Custom Modules	1237
SuiteScript 2.0 Entry Point Script Creation and Deployment	1240
Record-Level and Form-Level Scripts	1240
SuiteScript 2.0 Entry Point Script Validation	1241
Entry Point Script Validation Guidelines	1241
Entry Point Script Validation Examples	1243
Entry Point Script Validation Error Reference	1246
SuiteScript 2.0 Record-Level Scripts	1249
Script Record Creation	1250
Script Deployment	1253
Viewing System Notes	1256
SuiteScript 2.0 Form-Level Scripts	1257
Attaching a Client Script to a Form	1257
Configuring a Custom Action	1258
SuiteScript 2.0 Custom Modules	1260
SuiteScript 2.0 Anatomy of a Custom Module Script	1261
SuiteScript 2.0 Custom Module Tutorial	1262
Module Dependency Paths	1273
Naming a Custom Module	1274
Custom Module Examples	1275
Troubleshooting Errors	1279
Frequently Asked Questions: Custom Modules	1279
SuiteScript 2.0 Scripting Records and Subrecords	1281
SuiteScript 2.0 Scripting Records	1281
SuiteScript 2.0 – Standard and Dynamic Modes	1281
SuiteScript 2.0 – Record Modules	1282
SuiteScript 2.0 Scripting Subrecords	1282
Understanding Subrecords	1283
Subrecord Scripting in SuiteScript 2.0 Compared With 1.0	1298
Scripting Subrecords that Occur on Sublist Lines	1299
Scripting Subrecords that Occur in Body Fields	1323
SuiteScript 2.0 Custom Pages	1345
SuiteScript 2.0 Custom Forms	1345
Sample Custom Form Script	1351
SuiteScript 2.0 Custom List Pages	1366
Sample Custom List Page Script	1367

SuiteScript 2.0 API Introduction

- [SuiteScript 2.0 Hello World](#)
- [SuiteScript 2.0 Script Basics](#)
- [SuiteScript 2.0 Anatomy of a Script](#)
- [SuiteScript 2.0 Script Creation Process](#)
- [SuiteScript 2.0 Advantages](#)
- [SuiteScript 2.0 Terminology](#)
- [SuiteScript 2.0 Developer Resources](#)
- [SuiteScript Versioning Guidelines](#)

SuiteScript 2.0 Hello World

SuiteScript 2.0 is a JavaScript API that offers a broad range of options for enhancing and extending NetSuite. You can use SuiteScript to customize the behavior of a page, create custom workflows, schedule tasks, and much more.

To help you understand how SuiteScript 2.0 works, this topic walks you through the implementation of a basic customization. After you complete the steps in this topic, the system displays a “Hello, World!” message whenever you load a NetSuite task record.

This topic contains the following sections:

- [Key Concepts](#)
- [Step One: Enable the Feature](#)
- [Step Two: Create the Script File](#)
- [Step Three: Upload the Script File to NetSuite](#)
- [Step Four: Create a Script Record and Script Deployment Record](#)
- [Step Five: Test the Script](#)
- [Next Steps](#)

Key Concepts

Before proceeding with the steps in this topic, it may be useful to consider a few concepts and terms that are central to SuiteScript 2.0 development. Some of these terms are referenced in the steps that appear later in this topic. You can read about these concepts now, or you can go straight to [Step One: Enable the Feature](#).

SuiteScript 2.0 Script Types and Entry Points

If you have written scripts before, you are probably used to thinking through basic questions prior to coding. These questions might include: What does the script need to accomplish? Should the script run in the browser or on the server? What specific event should trigger the script?

With SuiteScript 2.0, part of this process involves deciding which of the system’s predefined **script types** you should use. Each script type is designed for a specific type of situation and specific types of triggering events. The following are some of the available [SuiteScript 2.0 Script Types](#):

- The [SuiteScript 2.0 Client Script Type](#) is designed for scripts that should run in the browser.

- The [SuiteScript 2.0 Scheduled Script Type](#) is for server-side scripts that should run at a specific time or on a recurring schedule.
- The [SuiteScript 2.0 RESTlet Script Type](#) is for server-side scripts that should execute when called over HTTP by an application external to NetSuite.

Each script type includes one or more **entry points** that are exclusive to that type. The entry point represents the juncture at which the system grants control of the NetSuite application to the script. When you include an entry point in your script, you tell the system that it should do something when that entry point is invoked. Specifically, you tell the system that it should execute a function defined within the script. This function is called an **entry point function**.

With many script types, the available entry points are analogous to types of events — various things that can happen — to trigger your script. For example, the client script type's entry points represent events that can occur during a browser session. These entry points include [fieldChanged](#), which represents a change to the value of a field, and [pagenit](#), which represents the loading of a page. In comparison, the scheduled script type has only one entry point, called [execute](#). It represents the point at which a schedule executes the script or a user acts manually to execute the script.

In the example used in this topic, you want a dialog alert to appear when a user working in a browser loads the NetSuite task record page. For that reason, this example uses the client script type and the [pagenit](#) entry point.

SuiteScript 2.0 Modules

SuiteScript 2.0 has a [Modular Architecture](#). As one indication of this, all SuiteScript 2.0 APIs are organized into a series of standard modules. Each module's name reflects its functionality. For example, the [N/record Module](#) lets you interact with NetSuite records. The [N/https Module](#) lets you make https requests to external web services.

Most modules must be explicitly loaded by a script before the script can access that module's APIs. At a very high level, loading a JavaScript module is similar to importing a library in Java. It is a way of providing access to logic that is defined elsewhere. In an entry point script, you load a module by using the [define Object](#). You list the modules that you want to load as an argument of the define function.

In contrast, some APIs are globally available. When an object, method, or function is globally available, it can be used even when you do not explicitly load the module to which it belongs. Globally available APIs are listed in [SuiteScript 2.0 Global Objects](#).

The example in this topic uses both approaches: It uses globally available APIs. It also uses a method that becomes available only after the appropriate module is loaded.

Entry Point Scripts Versus Custom Module Scripts

The example script in this topic is relatively simple. All of its logic is contained within one script file. However, you might want to create scripts that rely on logic defined in other script files. In SuiteScript 2.0, these supporting script files are known as **custom module scripts**.

In contrast, the primary script file — the one that identifies the script type, entry point, and entry point function — is known as an **entry point script**. The system imposes formatting requirements on entry point scripts that are different from those of custom module scripts. The remaining steps in this topic highlight some of the requirements that exist for entry point scripts.

Custom module scripts are not covered in this topic. For information about custom module scripts, see [SuiteScript 2.0 Custom Modules](#).

Step One: Enable the Feature

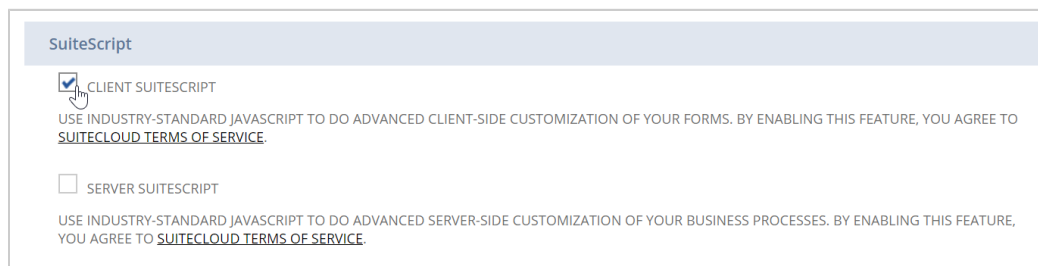
Before you can complete the rest of the steps in this topic, the Client SuiteScript feature must be enabled in your NetSuite account.

If you are not sure whether the feature is enabled, you can check by navigating to Customization > Scripting > Scripts. If you can access that path, the feature is enabled. If the option does not appear, the reason could be that you do not have permission to access it, or that the menu path has been customized. If you are not sure, check with your account administrator.

The feature can be enabled by an account administrator or by a user with the Enable Features permission. To enable the feature, an authorized user should complete the following steps.

To enable the Client SuiteScript feature:

1. Select Setup > Company > Enable Features.
2. Click the **SuiteCloud** subtab.
3. Locate the **Client SuiteScript** option. If it the box is already checked, skip ahead to [Step Two: Create the Script File](#). If it is not, check the box.



The screenshot shows a window titled "SuiteScript". It contains two sections. The first section, "CLIENT SUITESCRIPT", has a checked checkbox and text stating: "USE INDUSTRY-STANDARD JAVASCRIPT TO DO ADVANCED CLIENT-SIDE CUSTOMIZATION OF YOUR FORMS. BY ENABLING THIS FEATURE, YOU AGREE TO [SUITECLOUD TERMS OF SERVICE](#)." The second section, "SERVER SUITESCRIPT", has an unchecked checkbox and text stating: "USE INDUSTRY-STANDARD JAVASCRIPT TO DO ADVANCED SERVER-SIDE CUSTOMIZATION OF YOUR BUSINESS PROCESSES. BY ENABLING THIS FEATURE, YOU AGREE TO [SUITECLOUD TERMS OF SERVICE](#)."

The system displays a window listing the terms of service.

4. If you agree to the terms, scroll to the bottom of the window and click **I Agree**.
5. Server SuiteScript is not required for the steps described in this topic. However, if you plan to do further SuiteScript development, consider checking the **Server SuiteScript** box. If you check the box, the system displays another window listing the terms of service. Click **I Agree**.
6. Click **Save**.

Step Two: Create the Script File

Before proceeding, you must create a script file called helloWorld.js. To create this file, you can use either of the following approaches:

- If you want to copy and paste the completed script directly from this document, skip ahead to [Copy the Full Script](#).
- If you want to read about how to construct the script yourself, refer to [Create the Script Step by Step](#).

Create the Script Step by Step

The following steps walk you through the process of creating the helloWorld.js sample script. Many of the steps in this process are required for any entry point script.

To create the script file:

1. Open a new file in your text editor of choice.
2. Add two JSDoc tags, **@NApiVersion** and **@NScriptType**. After each tag, add the appropriate value, as shown in the following snippet.

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

```

These tags reflect the following:

- The version of SuiteScript you are using.
- The script type you are using.

Every SuiteScript 2.0 entry point script must include these tags. For each tag, you can include only one value. For more information and a list of valid values, see [SuiteScript 2.0 JSDoc Tags](#).

3. Add the [define](#) **Object**. Every entry point script must use this function.

Use `['N/ui/dialog']` as the define function's first argument. This first argument is an array of string values representing the modules that the function should load.

```
...
define(['N/ui/dialog'],

    // In Step 4, you put additional code here.

);

```

The [N/ui/dialog Module](#) includes methods that display various types of dialogs. You load this module so that the script can use one of these methods.

4. Declare a callback function. The callback function is the define function's second argument. Give this function one argument called **dialog**.

```
...
function(dialog) {

    // In Step 5, you put additional code here.

}
...

```

If you are not familiar with callback functions, just remember that this function will contain all of the script's other logic. Additionally, remember that the number of arguments used by the callback function must equal the number of modules loaded by the define function. Each argument is an object that lets you access the module it represents. As a best practice, give each argument a name similar to that of the corresponding module.

In this example, the define function loads only one module, so the callback function has only one argument.

5. Within the callback function, declare a function called **helloWorld**.

```
...
function helloWorld() {

    // In steps 6-10, you put additional code here.

}

```

```
...
```

Later, you designate the `helloWorld` function as the script's entry point function. Every entry point script must have an entry point function.

A function is considered an entry point function only if it is linked to an entry point. You create this link in Step 11.

6. Within the entry point function, create an object named **options**.

Many SuiteScript 2.0 methods either require or can accept a plain JavaScript object as their argument. The method that you use to create the "Hello, World!" dialog falls into this category. This method accepts an object that has two parameters: `title` and `message`.

```
...
    var options = {
        title: 'Hello!',
        message: 'Hello, World!'
    };
...
```

7. Add a [try/catch statement](#). This statement is not required. However, this approach lets your script handle errors gracefully. That is, if an error occurs and is handled by a try/catch statement, your script — and any others that are deployed on the page — can continue executing. Using a try/catch statement can also help you understand why problems occur. Note that the try/catch keywords are part of JavaScript and not specific to SuiteScript 2.0.

A basic try/catch statement consists of two parts. The try block holds code that you want to execute. The catch block holds logic that should execute if JavaScript errors are encountered during the try block.

Add the try/catch statement after the object that you created in Step 6.

```
...
    try {

        // In steps 8 and 9, you put additional code here.

    } catch (e) {

        // In Step 10, you put additional code here.

    }
...
```

8. For the first action of the try block, invoke the [N/ui/dialog Module's alert\(\)](#) method. This method creates a dialog with a title, a message, and an OK button.

You invoke this method by using the object which, in Step 4, you named **dialog**. You also use the method's name, **alert**. To define the dialog's title and message, pass in the object titled **options**.

```
...
        dialog.alert(options);
...
```

For more details about this method, see the [dialog.alert\(options\)](#) reference page. Note that the title of the [dialog.alert\(options\)](#) reference page matches the code you have added to your script. However, be aware that the reference pages for standard module methods may not always reflect your naming conventions. For example, if you had specified an argument named

message when you declared your callback function, you would invoke the alert method using the expression `message.alert(options)`.

Similarly, note that within the documentation, each method's argument is typically referenced as an object titled `options`. However, in your script, you can give the object any name. You can also create the object directly, as part of the process of invoking the method.

9. Add logic to create a log entry when the dialog displays successfully. To do this, use the globally available `log.debug(options)` method.

The `debug()` method is called on the `log` object. Unlike the `dialog` object, which you had to take steps to access, the `log` object is made available to every script. For that reason, you don't give this object a name, so you can always write `log.debug` to call this method.

This method takes an object that has two properties: a title and a detailed message. This time, create the object directly. Contrast this style with the way you created an object in Step 5, then passed that object by name to the `alert()` method.

```
...
    log.debug ({
        title: 'Success',
        details: 'Alert displayed successfully'
    });
...
```

The log entry is created in the UI when a user triggers the dialog alert. An explanation of how to find the log is covered in [Step Five: Test the Script](#).

10. In the catch block, add logic to create a log entry if an error is thrown. Use the globally available `log.error(options)` method. The `log.error()` method is similar to the `log.debug()` method. The only difference is that, with `log.error()`, the log entry is classified as an entry of type error, instead of debug.

```
...
    log.error ({
        title: e.name,
        details: e.message
    });
...
```

11. Immediately after the entry point function, add a return statement. In every SuiteScript 2.0 entry point script, the return statement must include at least one line that has two components:

- An entry point, which in this case is `pageInit`.
- A function, which in this case is `helloWorld`.

```
...
    return {
        pageInit: helloWorld
    };
...
```

Because of this reference, `helloWorld` is considered an entry point function.

Although this script uses only one entry point, a return statement can include multiple entry points. Using multiple entry points is permitted as long as they all belong to the script type identified by the `@NScriptType` tag at the top of the file. For example, in addition to `pageInit`, the client script type also includes `saveRecord`. So, if you wanted the script to take one action when the page loads and another action when the user clicks Save, you could use both entry

points. For an example of a script that uses multiple entry points, see [SuiteScript Client Script Sample](#).

12. Save the file, naming it helloWorld.js.

Copy the Full Script

This section shows the full sample script. If you haven't already created the script file by using the steps described in [Create the Script Step by Step](#), copy and paste the following code into a text file. Save the file and name it helloWorld.js.

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

define(['N/ui/dialog'],

    function(dialog) {

        function helloWorld() {

            var options = {
                title: 'Hello!',
                message: 'Hello, World!'
            };

            try {

                dialog.alert(options);

                log.debug ({
                    title: 'Success',
                    details: 'Alert displayed successfully'
                });

            } catch (e) {

                log.error ({
                    title: e.name,
                    details: e.message
                });
            }

            return {
                pageInit: helloWorld
            };
        });
    });
```

 **Note:** If the script sample splits across multiple pages, make sure that you copy all of the code.

Step Three: Upload the Script File to NetSuite

After you have created your entry point script file, upload it to your NetSuite File Cabinet.

To upload the script file:

1. In the NetSuite UI, go to Documents > File > SuiteScripts.
2. Click **Add File**.
3. Follow the prompts to locate the helloWorld.js file in your local environment and upload it.

Be aware that even after you upload the file, you can edit it from within the File Cabinet, if needed. For details, see the help topic [Editing Files in the File Cabinet](#).

Step Four: Create a Script Record and Script Deployment Record

In general, before an entry point script can execute in your account, you must create a script record that represents the entry point script file. You must also create a script deployment record.

The script deployment record contains part of the logic that determines when the script executes. Some of that logic is contained within the script, by the entry point that the script uses. For example, by using the [pageInit](#) entry point, you tell the system that the script should execute when a page loads. However, you must also identify the specific page which, when loaded, causes the script to execute. Put another way, you must tell the system which record type this script should execute on. To do that, you use the script deployment record.

These records can be created programmatically. You can also create them in the UI, as described in the following procedure.

To create the script record and script deployment record:

1. Go to Customization > Scripting > Scripts > New.
2. In the **Script File** dropdown list, select helloWorld.js.
Note that, if you had not yet uploaded the file, as described in Step Three, you could upload the file from this page. With your cursor, point to the right of the dropdown list to display a plus icon. Clicking this icon opens a window that lets you upload a file.
3. After you have populated the dropdown list, click the **Create Script Record** button.
In response, the system displays a new script record, with the helloWorld.js file listed on the **Scripts** subtab.
4. Fill out the required body fields as follows:
 - In the **Name** field, enter **Hello World Client Script**.
 - In the **ID** field, enter **_cs_helloworld**.
5. Click the **Deployments** subtab. You use this subtab to create the deployment record.
6. Add a line to the sublist, as follows:
 - Set the **Applies to** dropdown list to **Task**. (Note that if you wanted the dialog to appear when a different type of record loads, you could select another record type from the list.)
 - In the **ID** field, enter **_cs_helloworld**.

Leave the other fields set to their default values. Note that the **Status** field is set to **Testing**. This value means that the script does not deploy for other users. (If you wanted to change the deployment later and make this customization available to all users, you could edit the deployment and set the status to **Released**.)

7. Click **Save**.

The system creates the script and script deployment records.

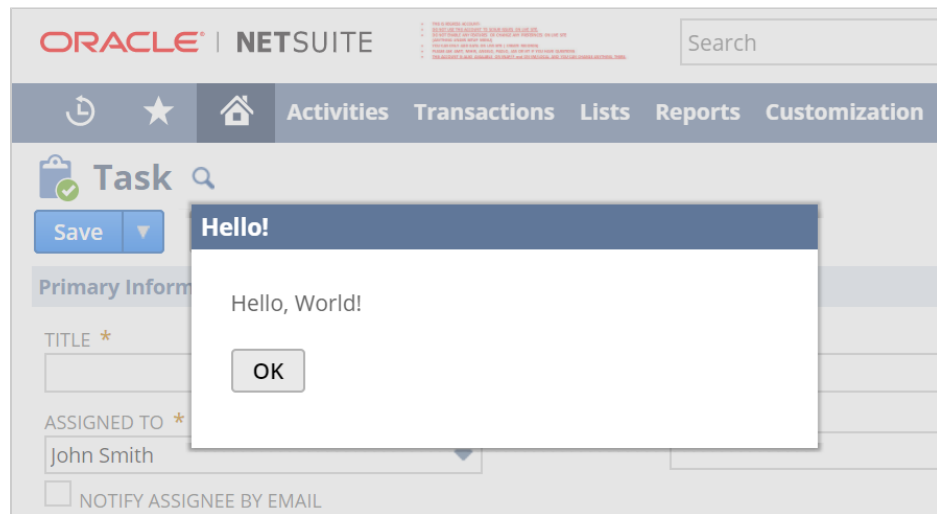
Step Five: Test the Script

Now that the script is deployed, you should verify that it executes as expected.

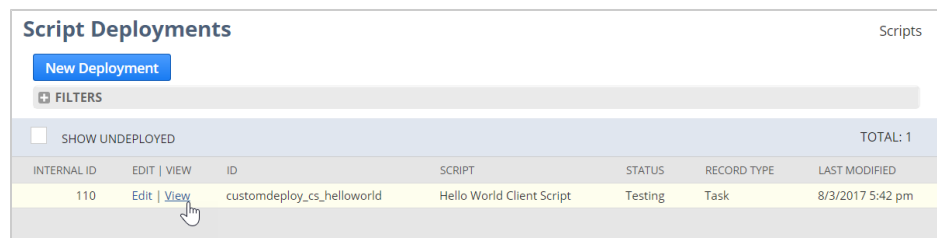
To test the script:

1. Verify that the dialog alert appears when it should:
 - a. Open a task record by going to Activities > Scheduling > Tasks > New.

If the script is working properly, the system displays a dialog alert.

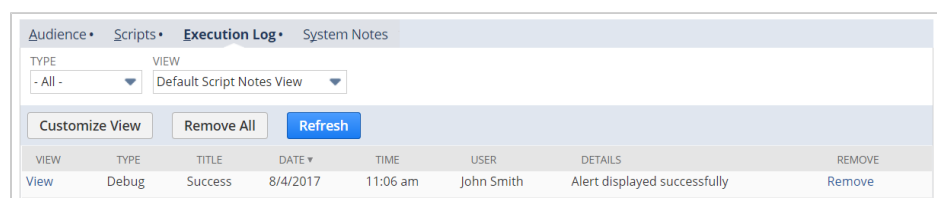


- b. Confirm and close the dialog by clicking **OK**.
2. Verify that the expected log entry has been saved to the script deployment record:
 - a. Go to Customization > Scripting > Script Deployments.
 - b. Locate your deployment, and click the corresponding **View** link.



- c. Click the **Execution Log** subtab.

The subtab should show an entry similar to the following.



The same entry also appears on the Execution Log of the script record.

If an error had been encountered, the error log entry you created would be displayed instead of the debug entry.

Next Steps

Now that you have deployed your first script, consider browsing other topics in [SuiteScript 2.0 API Introduction](#). Or, if you want to experiment with other script samples, try the following:

- Another commonly used script type is the user event type. This type is designed for server-side scripts that should execute when users take certain actions with records. To try a simple user event script, see [SuiteScript 2.0 User Event Script Tutorial](#).
- For a slightly more complex user event script, see [SuiteScript 2.0 User Event Script Sample](#).
- For an example of a custom module script, and a user event script that calls a custom module function, see [SuiteScript 2.0 Custom Module Tutorial](#).

SuiteScript 2.0 Script Basics

Certain components are common to all SuiteScript 2.0 scripts. This topic describes some of these components. It includes the following sections:

- [Modular Architecture](#)
- [Objects as Arguments](#)

 **Note:** If you have not already done so, review [SuiteScript 2.0 Hello World](#), paying particular attention to the [Key Concepts](#) section.

Modular Architecture

SuiteScript 2.0 is modular. As one indication of this, all of its APIs are organized into a series of standard modules. After your script loads a standard module, it can use any of that module's APIs.

But modularity has other implications, too. For example, each SuiteScript 2.0 entry point and custom module script must be structured so that it creates a module. If you have used SuiteScript 2.0 before, you may not have realized that your scripts were creating modules. However, every SuiteScript 2.0 script, with the exception of scripts intended solely for on-demand debugging in the NetSuite Debugger or a browser console, must create a module. This guideline affects both entry point scripts and custom module scripts. For example, an entry point script must create a module that can be used by the NetSuite application. A custom module script must create a module that can be loaded and used by other scripts. In both cases, the module must be designed to return an object, which is usually a function.

You create a module by using the globally available [define Object](#). If you have read [SuiteScript 2.0 Hello World](#), then you may recall that you can use the define function to load modules for use within your script. The ability to load modules is a significant part of the define function. However, an equally important aspect of the define function is its ability to create modules.

The define function can take between one and three arguments. However, a common technique, and the one used in most samples throughout this help center, is to provide two arguments, as follows:

- The first argument is a list of any dependencies, such as standard [SuiteScript 2.0 Modules](#). These modules are loaded when the script executes.
- The second argument is a callback function. The callback function must include a return statement that identifies at least one standard or custom entry point. The entry point must be associated with an object that is returned when the entry point is invoked. In most use cases, this object is a function.

If you want to understand more about the modular architecture of SuiteScript 2.0, consider reviewing the [Asynchronous Module Definition \(AMD\) Specification](#), which SuiteScript 2.0 uses. However, you do not have to be deeply familiar with AMD before you can use SuiteScript 2.0.



Note: Some samples that appear in this help center use the [require Function](#) instead of the `define` function. Samples that use `require` are intended for on-demand debugging, either in the NetSuite Debugger or in a browser console. If a script is to be used for anything other than on-demand debugging, it must use the [define Object](#).

Objects as Arguments

In SuiteScript 2.0, the arguments passed to methods are typically objects. For two examples of how this characteristic is significant, see the following sections:

- [Objects as Arguments for Standard Methods](#)
- [Context Objects Passed to Standard and Custom Entry Points](#)

Objects as Arguments for Standard Methods

Many SuiteScript 2.0 methods require you to provide one or more pieces of information. But regardless of how many discrete pieces of information are needed, generally all methods have one thing in common: they expect all input to be provided in the form of a single object. This object can have one property or it can have many.

For example, when you set a value on a record, you use the [Record.setValue\(options\)](#) method. With this method, you must submit two pieces of information: the ID of the field, and the value that you want to set for that field. To pass this information to the method, you must create an object with a **fieldId** property and a **value** property. Then you pass this object to the method. For example:

```
var myObject = {
    fieldId: 'Hello!',
    value: 'Hello, World!'
};

myRecord.setValue(myObject);
```

Another way of passing in an object is to define it within the method call. For example:

```
myRecord.setValue({
    fieldId: 'Hello!',
    value: 'Hello, World!'
});
```

In the documentation for standard SuiteScript 2.0 methods, each method's argument is typically referenced as an object titled **options**. However, in your script, you can give the object any name. For an example, see [SuiteScript 2.0 User Event Script Tutorial](#).

Context Objects Passed to Standard and Custom Entry Points

Every SuiteScript 2.0 script — every entry point script and every custom module script — must use an entry point. Further, each entry point must correspond with an object defined within the script. This object is usually a function. When the object is a function, it is referred to as an entry point function.

In most cases, an entry point function has access to an object provided by the system. Typically, this object can let your script access data and take actions specific to the context in which the script is executing. For that reason, this object is often referred to as a context object.

Depending on what your script needs to do, this object can be a critical part of your script. You can give this object any name you want, but most examples in this help center call these objects **context**.

Note: For an explanation of entry points, see the [Key Concepts](#) section of the [SuiteScript 2.0 Hello World](#).

Context Objects in Entry Point Scripts

Every standard script type includes entry points specific to that type. Most of these standard entry points have access to a context object that provides access to data or methods. The properties of this object vary depending on the entry point being used. For details about the context object available to each entry point, refer to the documentation for that entry point.

For an example of context objects being used in a user event script, see [SuiteScript 2.0 User Event Script Tutorial](#).

Context Objects in Custom Module Scripts

Just like an entry point script, in a custom module script, an entry point function can receive a context object. However, in the case of a custom module, the object is not provided by the NetSuite application. The object is provided by the SuiteScript that is calling the module.

If you want a custom entry point function to receive an object, then the SuiteScript that calls the function must be written in a way that supports that behavior: The calling script must create the object, set whatever properties are needed, and pass the object to the custom module script. The custom module can then use the object.

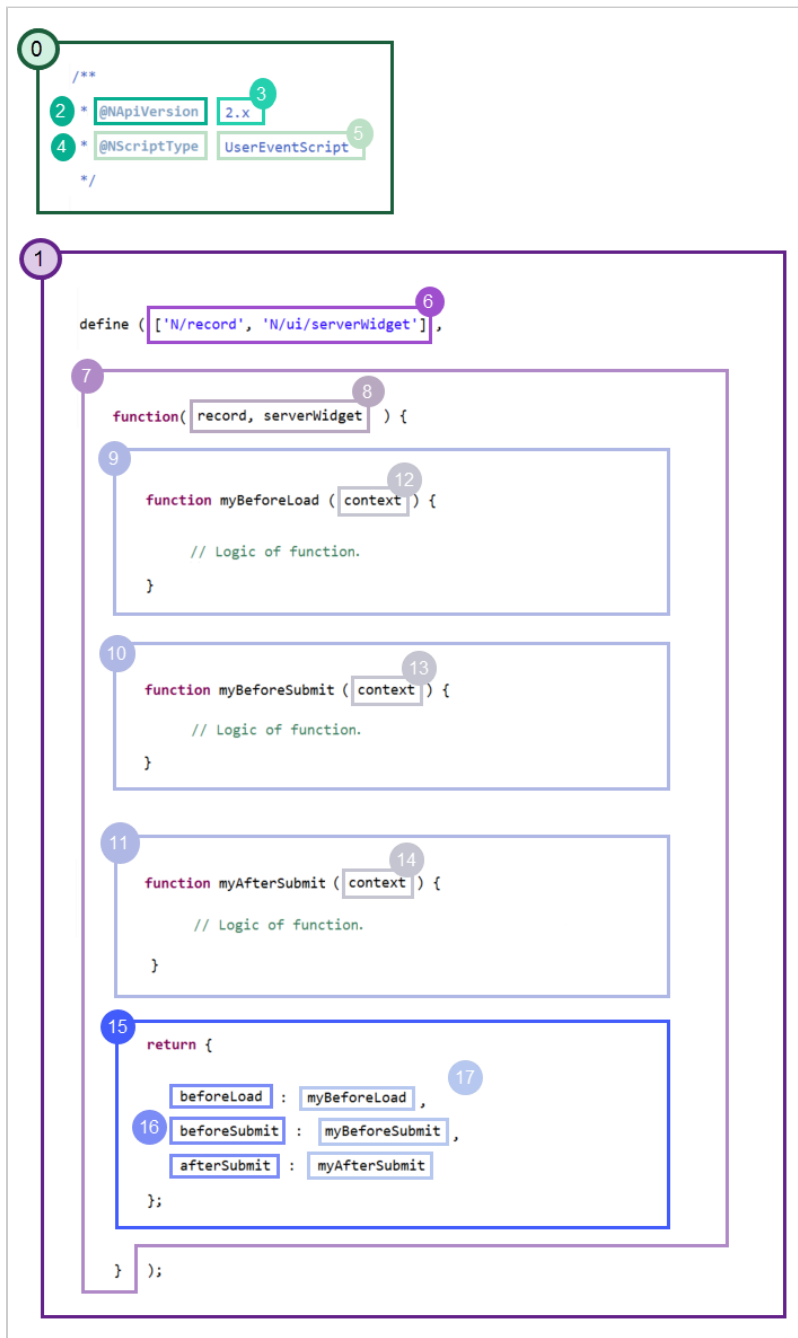
For an example, see [SuiteScript 2.0 Custom Module Tutorial](#).

Note: To see an example of how the components described in this topic appear in scripts, review [SuiteScript 2.0 Anatomy of a Script](#) and [SuiteScript 2.0 Anatomy of a Custom Module Script](#).

SuiteScript 2.0 Anatomy of a Script

All SuiteScript 2.0 entry point scripts must conform to the same basic structure. The following diagram illustrates that structure. For an explanation of the numbered components of the script, refer to the table that follows the diagram.

Note: For help with the terms defined in this topic, review [SuiteScript 2.0 Hello World](#).



General Area	Callout	Description
0 — JSDoc tags	2 and 3	The <code>@NapiVersion</code> tag, which is required in an entry point script, and its value. Valid values are 2.0, 2.x, and 2.X.
	4 and 5	The <code>@NScriptType</code> tag, which is required in an entry point script, and its value. The value is not case sensitive, but using Pascal case, as shown in this example, is recommended for better readability.

General Area	Callout	Description
1 — define statement	6	The define function's first argument, which is a list of dependencies, or a list of modules that the script loads. This script uses the N/record Module , which lets the script interact with records, and the N/ui/serverWidget Module , which lets the script interact with forms.
	7	The define function's second argument, which is a callback function.
	8	The arguments of the callback function. The first argument is an object that represents the N/record Module . The second represents the N/ui/serverWidget Module . The sequence of these objects matches the sequence of the define function's list of dependencies (Callout 6). These objects can be used anywhere in the callback function to access the APIs of those modules. You can give these objects any names you prefer. As a best practice, use names that are similar to the module names.
	9, 10, and 11	Entry point functions. For any function to be used, it must be named in the return statement alongside an entry point, as shown in Callout 17.
	12, 13, and 14	The context object provided to each entry point function. The characteristics of these objects vary depending on the entry point. For an explanation of these objects, see Context Objects Passed to Standard and Custom Entry Points .
	15	The callback function's return statement.
	16	The entry points used by the script. At least one entry point must be used. In an entry point script, any entry points used must belong to the script type identified by the @NScriptType tag (callouts 4 and 5).
	17	References to the script's entry point functions. For each entry point used, your script must identify an entry point function defined elsewhere in the script.

SuiteScript 2.0 Script Creation Process

The following is a very basic process flow for SuiteScript 2.0 script creation. For a more detail explanation with a sample script, see [SuiteScript 2.0 Hello World](#).

 **Note:** Your specific process may vary, depending on the content of your script.

Stage	Description	Additional Information
1	Use the <code>define()</code> function to load SuiteScript 2.0 modules in your entry point script. Your entry point script is the script you attach to the script record.	Modular Architecture SuiteScript 2.0 Global Objects
2	Add required JSDoc tags to your entry point script.	SuiteScript 2.0 JSDoc Validation
3	Add at least one entry point function to your entry point script. An entry point function is a named function that is executed when an entry point is triggered.  Important: Your entry point script can implement only one script type. For example, your entry point script cannot return both a <code>beforeLoad</code> entry point and an <code>onRequest</code> entry point.	SuiteScript 2.0 Script Types
4	Organize your supporting code into custom modules (as a replacement for SuiteScript 1.0 libraries). Create these modules with the <code>define()</code> function and then load them in your entry point script.	SuiteScript 2.0 Global Objects

Stage	Description	Additional Information
5	Upload and deploy your script to NetSuite.	SuiteScript 2.0 Entry Point Script Creation and Deployment

SuiteScript 2.0 Advantages

SuiteScript 2.0 is a complete redesign of the SuiteScript model. This topic goes over several of the advantages SuiteScript 2.0 has over SuiteScript 1.0.

- Complexity Management and Intuitive Code Organization
- Automatic Dependency Management
- Modern Programming Syntax and Behavior
- Functionality Enhancements

Complexity Management and Intuitive Code Organization

SuiteScript 2.0 is built on modularity. Modern SuiteApps require complex scripts that typically consist of many lines of code and many files. Modularity gives users built-in complexity management. It also adds encapsulation, provides intuitive code organization, and ensures there are no global variable or method naming conflicts.

SuiteScript 2.0 comes with a complete set of new APIs, contained within modules. These modules are organized and named based on behavior. For example, you use the file module when you need to work with files in NetSuite. Your script loads only those modules that it needs. With SuiteScript 1.0, all APIs are contained in the same global namespace. Each SuiteScript 1.0 script utilizes the entire namespace, regardless of which APIs it actually uses.

SuiteScript 2.0 also enables you to create your own custom modules. You can use these custom modules to organize your code (as a replacement for SuiteScript 1.0 libraries). Additionally, you can add custom modules to SuiteApps and expose those modules to third parties.

For additional information, see the following topics:

- SuiteScript 2.0 Hello World
- SuiteScript 2.0 Script Basics
- SuiteScript 2.0 Modules
- SuiteScript 2.0 Entry Point Script Creation and Deployment
- SuiteScript 2.0 Custom Modules



Note: SuiteScript 2.0 implements the Asynchronous Module Definition (AMD) specification. AMD is used to define and load JavaScript modules and their dependencies. For additional information regarding AMD, see <http://requirejs.org/docs/whyamd.html> and <https://github.com/amdjs/amdjs-api/blob/master/AMD.md>

Automatic Dependency Management

SuiteScript 2.0 gives you built-in dependency management. With SuiteScript 2.0, you define the SuiteScript 2.0 modules and custom modules that must load prior to script execution. The module loader automatically loads the dependencies of those modules, the dependencies of the

dependencies, and so forth. Automatic dependency management enables you to concentrate on logic instead of dependencies and load order.

For additional information, see the following topics:

- [SuiteScript 2.0 Hello World](#)
- [SuiteScript 2.0 Script Basics](#)
- [SuiteScript 2.0 Entry Point Script Creation and Deployment](#)
- [SuiteScript 2.0 Custom Modules](#)

Modern Programming Syntax and Behavior

The underlying design principle of SuiteScript 2.0 is that SuiteScript 2.0 === JavaScript. This results in a decreased learning curve for experienced developers. The syntax is straightforward JavaScript. And unlike SuiteScript 1.0, the behavior is consistent. Enhancements to syntax and behavior include the following:

- **Third party JavaScript API support:** SuiteScript 2.0 is designed to support all standard JavaScript. The supplied SuiteScript 2.0 APIs give you programmatic access to NetSuite functionality. For generic logic, use custom modules to load your preferred third party JavaScript APIs.
- **SuiteScript 1.0 nlapi/nlobj prefix retirement:** SuiteScript 2.0 is modeled to look and behave like modern JavaScript. To meet that objective, SuiteScript 2.0 methods and objects are not prefixed with nlapi and nlobj. This change also reflects the modular organization of SuiteScript 2.0. SuiteScript 1.0 methods and objects respectively belong to the nlapi and nlobj namespaces. SuiteScript 2.0 methods and objects are encapsulated within various modules.
- **Usage of properties and enumerations:** SuiteScript 2.0 adopts the usage of properties and enumerations. Most SuiteScript 1.0 getter and setter methods are replaced with properties. Enumerations encapsulate common constants (for example, standard record types).

 **Note:** JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation uses the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.

- **Updated sublist and column indexing:** The standard practice in the development world is to start indexing at 0. This behavior is observed in the majority of programming languages. SuiteScript 1.0 starts sublist and column indexing at 1. To bring SuiteScript into alignment with modern JavaScript, sublist and column indexing within SuiteScript 2.0 begins at 0.

For additional information, see the following topics:

- [SuiteScript 2.0 Hello World](#)
- [SuiteScript 2.0 Entry Point Script Creation and Deployment](#)
- [SuiteScript 2.0 Custom Modules](#)

Functionality Enhancements

The following enhancements are exclusive to SuiteScript 2.0:

- [Map/Reduce Script Type](#)
- [Asynchronous Client Side Processing \(Promises\)](#)
- [SFTP File Transfer API](#)
- [Cache API](#)

- [Search Pagination API](#)
- [Flat File Streaming API](#)
- [Expanded Support for HTTP Content Type Headers](#)
- [New Encryption/Encoding Functionality](#)

Map/Reduce Script Type

SuiteScript 2.0 introduces a new server-side script type based on the map/reduce model. Map/ reduce scripts provide a structured framework for server-side scripts that process a large number of records. In addition, SuiteCloud Plus users can also use map/reduce scripts to process records in parallel across multiple processors. Users manually select the number of processors to utilize from the script deployment record.

For additional information, see [SuiteScript 2.0 Map/Reduce Script Type](#).

Asynchronous Client Side Processing (Promises)

Promises are JavaScript objects that represent the eventual result of an asynchronous process. After these objects are created, they serve as placeholders for the future success or failure of an operation. During the period of time that a promise object is waiting in the background, the remaining segments of the script can execute.

In SuiteScript 2.0, all client scripts support the use of promises. With promises, developers can write asynchronous code that is intuitive and efficient. SuiteScript 2.0 provides promise APIs for selected modules. In addition, you can create custom promises in all client scripts.

For additional information see [Promise Object](#).

SFTP File Transfer API

SuiteScript 2.0 provides support for SFTP (Secure File Transfer Protocol). This feature enables you to securely transfer files between NetSuite and external FTP (File Transfer Protocol) servers. SFTP is a protocol packaged with SSH (Secure Shell). It is similar to FTP, but files are transferred over a secure connection. Server authorization requires a password GUID (Globally Unique Identifier) and a DSA (Digital Signature Algorithm), ECDSA (Elliptical Curve Digital Signature Algorithm), or RSA (cryptosystem) host key.

For additional information, see [N/sftp Module](#).

Cache API

The SuiteScript 2.0 Cache API enables you to load data into a cache and make it available to one or more scripts. This feature reduces the amount of time required to retrieve data.

For additional information, see [N/cache Module](#).

Search Pagination API

The SuiteScript 2.0 Search Pagination API enables you to page through search results. This enhancement increases script performance and gives you an intuitive means to efficiently traverse search result data.

For additional information, see [N/search Module](#).

Flat File Streaming API

With SuiteScript 1.0, you cannot easily access the contents of files that are over 10MB. You must partition your files into smaller, separate files to read, write, and append file contents in memory.

The SuiteScript 2.0 Flat File Streaming API enables you to efficiently process and stream large CSV and plain text files. With this enhancement, you can load and edit each line of content into memory, and then append the lines back together. The Flat File Streaming API enforces the 10MB limit only on individual lines of content.

For additional information, see [N/file Module](#).

Expanded Support for HTTP Content Type Headers

SuiteScript 2.0 adds support for all HTTP content types. This enhancement applies to both client request and server response HTTP headers.

For additional information, see [N/http Module](#) and [N/https Module](#).

New Encryption/Encoding Functionality

SuiteScript 2.0 adds enhanced encryption, decryption, and hashing functionality with the [N/crypto Module](#). Additional encoding functionality is exposed in the [N/encode Module](#).

SuiteScript 2.0 Terminology

Note: The definitions listed reflect the usage of the terms as they apply to SuiteScript 2.0. Some terms may have alternate meanings outside of the SuiteScript context.

Term	Definition
Custom Module	A custom module is a user-defined module that serves as a JavaScript library or supporting logic. This module is separate from your entry point script. Your entry point script loads this module as a dependency. For additional information on custom modules, see SuiteScript 2.0 Custom Modules .
Deferred Dynamic Mode	See the definition for standard mode.
Dynamic Mode	See the definition for standard mode.
Entry Point	An entry point represents the juncture at which the system grants control of the NetSuite application to the script. Each script type includes one or more entry points that are exclusive to that type. When that entry point is invoked, the system knows to execute its corresponding entry point function. For additional information on entry points, see SuiteScript 2.0 Entry Point Script Validation , SuiteScript 2.0 Hello World , and SuiteScript 2.0 Script Types .
Entry Point Function	A function that executes when an entry point is invoked. For additional information on entry point functions, see SuiteScript 2.0 Entry Point Script Validation and SuiteScript 2.0 Hello World .
Entry Point Script	Your entry point script is the primary script that you attach to the script record. This script identifies the script type, entry points, and entry point functions. Each entry point script must include at least one entry point and entry point function.

Term	Definition
	For additional information on entry point scripts, see SuiteScript 2.0 Entry Point Script Validation and SuiteScript 2.0 Hello World .
Enum	JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Governance	Governance ensures that an individual script does not consume an unreasonable amount of resources. The SuiteScript governance model is based on usage units. Certain API calls cost a specific number of usage units. Each script type has a maximum number of usage units that it can expend per execution. If the number of allowable usage units is exceeded, the script is terminated and an error is thrown. For additional information on governance, see the help topic SuiteScript Governance .
Script	A script refers to an aggregate of <ul style="list-style-type: none"> ■ An entry point script file ■ All custom modules used by the entry point script ■ The script record associated with the entry point script For additional information on the components that make up a script, see SuiteScript 2.0 Entry Point Script Validation , SuiteScript 2.0 Custom Modules , Script Record Creation , and SuiteScript 2.0 Hello World .
Script Deployment	A script deployment determines a portion of the associated script's runtime behavior. The settings you can define on a script deployment record include: <ul style="list-style-type: none"> ■ The record types the script executes against ■ When the script is executed ■ The audience and role restrictions ■ The script log levels The settings found on the deployment record vary based on script type. For additional information on script deployments, see SuiteScript 2.0 Entry Point Script Creation and Deployment and SuiteScript 2.0 Hello World .
Script Type	SuiteScript 2.0 scripts consist of several script types. Each script type is designed for a specific type of situation and specific types of triggering events. For additional information on script types, see SuiteScript 2.0 Script Types .
Standard Mode	Standard mode is also referred to as deferred dynamic mode. There are two modes you can operate in when you work with a record in SuiteScript: standard mode and dynamic mode. <ul style="list-style-type: none"> ■ In standard mode, the record's body fields and sublist line items are not sourced, calculated, and validated until the record is saved. ■ In dynamic mode, the record's body fields and sublist line items are sourced, calculated, and validated in real-time. A record in dynamic mode emulates the behavior of a record in the UI. For additional information on standard and dynamic mode, see record.Record .
Usage Units	See the definition for governance.

SuiteScript 2.0 Developer Resources

SuiteScript 2.0 – Help

See the following help sections for information about developing with SuiteScript 2.0:

- [SuiteScript 1.0 to SuiteScript 2.0 API Map](#)
- [SuiteScript 2.0 Script Types](#)
- [SuiteScript 2.0 Global Objects](#)
- [SuiteScript 2.0 Modules](#)
- [SuiteScript 2.0 JSDoc Validation](#)
- [SuiteScript 2.0 Entry Point Script Creation and Deployment](#)
- [SuiteScript 2.0 Custom Modules](#)
- [SuiteScript 2.0 Scripting Records and Subrecords](#)
- [SuiteScript 2.0 Custom Pages](#)

SuiteScript 2.0 – Internal Resources

The following internal resources are available in addition to the SuiteScript 2.0 help.

Resource	Description
NetSuite User Group	Official forum for the NetSuite community. Use the search field to find the SuiteScript 2.0 board.
SuiteScript 2.0: Extend NetSuite with JavaScript	Oracle Training Course
SuiteScript 2.0 for Experienced Developers	Oracle Training Course

SuiteScript 2.0 – External Resources

The following external resources are available in addition to the SuiteScript 2.0 help.

Resource	Description
AMD Specification	SuiteScript 2.0 implements its modular architecture with the Asynchronous Module Definition (AMD) specification. AMD is used to define and load JavaScript modules and their dependencies.
RequireJS	RequireJS also implements the AMD specification.
https://www.promisejs.org/	Tutorials on JavaScript promises
Eloquent JavaScript	Free online JavaScript book
You Don't Know JS	Free online JavaScript book series
Object-oriented JavaScript for Beginners	MDN article on object oriented JavaScript
StackOverflow – SuiteScript 2.0	Online community for developers

SuiteScript Versioning Guidelines

See the following sections for information about SuiteScript versioning:

- [SuiteScript Versioning](#)

■ Version Cohabitation Rules

SuiteScript Versioning

This release (SuiteScript 2.0) and all future releases of SuiteScript will maintain the following versioning system.

Version Type	Numbering Pattern	Description
Major	Version 2.0, 3.0, 4.0	Major versions of SuiteScript include significant functionality changes and improvements. Major versions are not backward compatible with previously released versions.
Minor	Version 3.1, 3.2, 3.3	Minor versions of SuiteScript include enhancements to existing features. Minor versions are backward compatible with all versions released since the last major version. For example, SuiteScript Version 3.2 is backward compatible with Versions 3.0 and 3.1. It is not backward compatible with Versions 1.0 or 2.0.
Patch	Does not apply	Patch versions of SuiteScript are included with regular NetSuite bug fix releases. Patch versions are backward compatible with all versions released since the last major version.

Version Cohabitation Rules

Your script (entry point script and supporting library scripts) must use either SuiteScript 1.0 or SuiteScript 2.0. You cannot use APIs from both versions in one script.

You can, however, have multiple scripts that use different SuiteScript versions. These can be deployed in the same account, in the same SuiteApp, and on the same record.

The map/reduce script type is a new functionality introduced with SuiteScript 2.0. You cannot use `nlapiScheduleScript(scriptId, deployId, params)` to schedule a Map/Reduce script. See [SuiteScript 2.0 Map/Reduce Script Type](#) for more information.

SuiteScript 1.0 to SuiteScript 2.0 API Map



Important: These topics are a work in progress. Some items are currently missing or do not have content. Additional updates are forthcoming.

These topics map SuiteScript 1.0 APIs to their corresponding SuiteScript 2.0 APIs. Keep the following in mind when using these mappings:

- Some SuiteScript 1.0 APIs do not have a SuiteScript 2.0 equivalent.
- There is not always a one to one mapping between SuiteScript 1.0 and SuiteScript 2.0. Each SuiteScript 1.0 API is listed only one time, but it may map to several SuiteScript 2.0 APIs.
- These mappings **do not include** SuiteScript 1.0 deprecated APIs.
- These mappings **do not include** new SuiteScript 2.0 functionality. To find new SuiteScript 2.0 functionality, go to [SuiteScript 2.0 Modules](#). The table includes a description of, and link to, each module.

These topics group SuiteScript 1.0 APIs into functions (prefixed with “nlapi”) and objects (prefixed with “nlobj”). All functions are listed alphabetically in one table. Whereas objects and their members are grouped alphabetically by object name. Each object has its own table containing all object members.

- [SuiteScript 1.0 to SuiteScript 2.0 API Map – Functions \(nlapi\)](#)
- [SuiteScript 1.0 to SuiteScript 2.0 API Map – Objects \(nlobj\)](#)

SuiteScript 1.0 to SuiteScript 2.0 API Map – Functions (nlapi)

This topic maps SuiteScript 1.0 Functions (prefixed with “nlapi”) to their corresponding SuiteScript 2.0 APIs. All functions are listed alphabetically in one table.



Note: To view a mapping of SuiteScript 1.0 Objects (prefixed with “nlobj”) to their corresponding SuiteScript 2.0 APIs, see [SuiteScript 1.0 to SuiteScript 2.0 API Map – Objects \(nlobj\)](#).

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiAddDays(d, days)	See Notes	See Notes	<p>This API does not have a SuiteScript 2.0 equivalent.</p> <p>Use the following JavaScript to add or subtract days from a Date object: <code>dateObj.setDate(dateObj.getDate() + or - days)</code></p> <p>For example:</p> <pre>var tomorrow = new Date(); tomorrow.setDate(tomorrow.getDate() + 1);</pre>

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			 Note: SuiteScript 2.0 is also compatible with third party JavaScript APIs that provide this functionality (for example, Moment.js). For information on using third party APIs with SuiteScript 2.0, see SuiteScript 2.0 Custom Modules .
nlapiAddMonths(d, months)	See Notes	See Notes	<p>This API does not have a SuiteScript 2.0 equivalent.</p> <p>Use the following JavaScript to add or subtract months from a Date object:</p> <pre>dateObj.setMonth(dateObj.getMonth() + or - months)</pre> <p>For example:</p> <pre>var today = new Date(); var onemonthago = today.setMonth(today.getMonth() - 1);</pre>  Note: SuiteScript 2.0 is also compatible with third party JavaScript APIs that provide this functionality (for example, Moment.js). For information on using third party APIs with SuiteScript 2.0, see SuiteScript 2.0 Custom Modules .
nlapiAttachRecord(type, id, type2, id2, attributes)	record.attach(options)	N/record Module	<pre>var recordId = record.attach({ record: { type: record.Type.FILE, id: '447' }, to: { type: record.Type.CUSTOMER, id: 530 } });</pre>
nlapiCancelLineItem(type)	Record.cancelLine(options) CurrentRecord.cancelLine(options)	N/record Module N/currentRecord Module	-
nlapiCommitLineItem(type)	Record.commitLine(options) CurrentRecord.commitLine(options)	N/record Module N/currentRecord Module	<p>For N/record script samples, see:</p> <ul style="list-style-type: none"> N/record Module Script Samples Example: Creating an Inventory Detail Sublist Subrecord
nlapiCopyRecord(type, id, initializeValues)	record.copy(options)	N/record Module	<pre>var recObj = record.copy({</pre>

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			<pre> type: record.Type.SALES_ORDER, id: 284, isDynamic: true, defaultValues: { entity: 547 }; var recordId = record.save(); </pre>
nlapiCreateAssistant(title, hideHeader)	serverWidget.createAssistant(options)	N/ui/serverWidget Module	-
nlapiCreateCSVImport()	task.create(options)	N/task Module	For script samples, see N/task Module Script Samples .
nlapiCreateCurrentLineItemSubrecord(sublist, fld name)	Record.getCurrentSublistSubrecord(options) CurrentRecord.getCurrentSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiCreateEmailMerger(templateId)	render.mergeEmail(options)	N/render Module	-
nlapiCreateError(code, details, suppressNotification)	error.create(options)	N/error Module	For a script sample, see N/error Module Script Samples .
nlapiCreateFile(name, type, contents)	file.create(options)	N/file Module	For a script sample, see N/file Module Script Sample .
nlapiCreateForm(title, hideNavbar)	serverWidget.createForm(options)	N/ui/serverWidget Module	For a script sample, see N/ui/serverWidget Module Script Samples
nlapiCreateList(title, hideNavbar)	serverWidget.createList(options)	N/ui/serverWidget Module	-
nlapiCreateRecord(type, initializeValues)	record.create(options)	N/record Module	-
nlapiCreateSearch(type, filters, columns)	search.create(options)	N/search Module	For a script sample, see N/search Module Script Samples .
nlapiCreateSubrecord(fld name)	Record.getSubrecord(options) CurrentRecord.getSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiCreateTemplateRenderer()	render.create()	N/render Module	For a script sample, see N/render Module Script Sample .
nlapiDateToString(d, format)	format.format(options)	N/format Module	For a script sample, see N/format Module Script Samples
nlapiDeleteFile(id)	file.delete(options)	N/file Module	-
nlapiDeleteRecord(type, id, initializeValues)	record.delete(options)	N/record Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiDetachRecord(type, id, type2, id2, attributes)	record.detach(options)	N/record Module	-
nlapiDisableField(fldnam, val)	Field.isDisabled	N/currentRecord Module	Note that <code>isDisabled</code> is a property.
nlapiDisableLineItemField(type, fldnam, val)	Field.isDisabled	N/currentRecord Module	Note that <code>isDisabled</code> is a property.
nlapiEditCurrentLineItemSubrecord(sublist, fldname)	Record.getCurrentSublistSubrecord(options) CurrentRecord.getCurrentSublistSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiEditSubrecord(fldname)	Record.getSubrecord(options) CurrentRecord.getSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiEncrypt(s, algorithm, key)	See Notes	See Notes	For SuiteScript 2.0 encryption, hashing, and HMAC functionality, see the N/crypto Module module. For SuiteScript 2.0 encoding functionality, see the N/encode Module module.
nlapiEscapeXML(text)	xml.escape(options)	N/xml Module	-
nlapiExchangeRate(sourceCurrency, targetCurrency, effectiveDate)	currency.exchangeRate(options)	N/currency Module	For a script sample, see N/currency Module Script Sample .
nlapiFindLineItemMatrixValue(type, fldnam, val, column)	Record.findMatrixSublistLineWithValue(options) CurrentRecord.findMatrixSublistLineWithValue(options)	N/record Module N/currentRecord Module	
nlapiFindLineItemValue(type, fldnam, val)	Record.findSublistLineWithValue(options) CurrentRecord.findSublistLineWithValue(options)	N/record Module N/currentRecord Module	-
nlapiFormatCurrency(str)	format.format(options)	N/format Module	Note that SuiteScript 2.0 currency for matting is handled by the N/format module and not the N/currency module . For a script sample, see N/format Module Script Samples
nlapiGetContext()	runtime.getCurrentScript() runtime.getCurrentSession() runtime.getCurrentUser()	N/runtime Module	For a script sample, see N/runtime Module Script Sample .
nlapiGetCurrentLineItemDateTimeValue(type, fieldId, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiGetCurrentLineItemIndex(type)	Record.getCurrentSublistIndex(options) CurrentRecord.getCurrentSublistIndex(options)	N/options Module N/ currentRecord Module	-
nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)	CurrentRecord.getCurrentMatrixValue(options) Record.getCurrentMatrixSublistValue(options)	N/currentRecord Module N/ currentRecord Module	
nlapiGetCurrentLineItemText(type, fldnam)	Record.getCurrentSublistText(options) CurrentRecord.getCurrentSublistText(options)	N/options Module N/ currentRecord Module	-
nlapiGetCurrentLineItemValue(type, fldnam)	Record.getCurrentSublistValue(options) CurrentRecord.getCurrentSublistValue(options)	N/options Module N/ currentRecord Module	-
nlapiGetCurrentLineItemValues(type, fldnam)	Record.getCurrentSublistValue(options) CurrentRecord.getCurrentSublistValue(options)	N/options Module N/ currentRecord Module	-
nlapiGetDateTimeValue(fldId, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
nlapiGetDepartment()	User.department	N/runtime Module	-
nlapiGetField(fldnam)	Record.getField(options) CurrentRecord.getField(options)	N/record Module N/ currentRecord Module	-
nlapiGetFieldText(fldnam)	Record.getText(options) CurrentRecord.getText(options)	N/record Module N/ currentRecord Module	-
nlapiGetFieldTexts(fldnam)	Record.getText(options) CurrentRecord.getText(options)	N/record Module N/ currentRecord Module	-
nlapiGetFieldValue(fldnam)	Record.getValue(options) CurrentRecord.getValue(options)	N/record Module N/ currentRecord Module	-
nlapiGetFieldValues(fldnam)	Record.getValue(options) CurrentRecord.getValue(options)	N/record Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
		N/ currentRecord Module	
nlapiGetJobManager(jobType)	task.create(options)	N/task Module	For a script sample, see N/task Module Script Samples.
nlapiGetLineItemCount(type)	Record.getLineCount(options) CurrentRecord.getLineCount(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemDateTimeValue(type, fieldId, lineNumber, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
nlapiGetLineItemField(type, fldnam, linenum)	Record.getSublistField(options) CurrentRecord.getSublistField(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemMatrixField(type, fldnam, linenum, column)	Record.getMatrixSublistField(options) CurrentRecord.getMatrixSublistField(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)	Record.getMatrixSublistValue(options) CurrentRecord.getMatrixSublistValue(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemText(type, fldnam, linenum)	Record.getSublistText(options) CurrentRecord.getSublistText(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemValue(type, fldnam, linenum)	Record.getSublistValue(options) CurrentRecord.getSublistValue(options)	N/record Module N/ currentRecord Module	-
nlapiGetLineItemValues(type, fldname, linenum)	Record.getSublistValue(options) CurrentRecord.getSublistValue(options)	N/record Module N/ currentRecord Module	Method returns an array for multi-select fields.
nlapiGetLocation()	User.location	N/runtime Module	Note that location is a property.
nlapiGetLogin()	auth.changeEmail(options) auth.changePassword(options)	N/auth Module	For a script sample, see N/auth Module Script Sample.
nlapiGetMatrixCount(type, fldnam)	Record.getMatrixHeaderCount(options) CurrentRecord.getMatrixHeaderCount(options)	N/record Module N/ currentRecord Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiGetMatrixField(type, fldnam, column)	Record.getMatrixHeaderField(options) CurrentRecord.getMatrixHeaderField(options)	N/record N/ currentRecord Module	-
nlapiGetMatrixValue(type, fldnam, column)	Record.getMatrixHeaderValue(options) CurrentRecord.getMatrixHeaderValue(options)	N/record N/ currentRecord Module	-
nlapiGetNewRecord()	See Notes	See Notes	<p>To mimic this functionality in SuiteScript 2.0, use the following code in a beforeLoad(scriptContext), beforeSubmit(scriptContext), or afterSubmit(scriptContext) user event script.</p> <pre>function afterSubmit(context) { var newRec = context.newRecord; }</pre> <p>For additional information and a full script sample, see SuiteScript 2.0 User Event Script Type</p>
nlapiGetOldRecord()	See Notes	See Notes	<p>To mimic this functionality in SuiteScript 2.0, use the following code in a beforeSubmit(scriptContext) or afterSubmit(scriptContext) user event script.</p> <pre>function afterSubmit(context) { var oldRec = context.oldRecord; }</pre> <p>For additional information and a full script sample, see SuiteScript 2.0 User Event Script Type</p>
nlapiGetRecordId()	Record.id CurrentRecord.id	N/record Module N/ currentRecord Module	-
nlapiGetRecordType()	Record.type CurrentRecord.type	N/record Module N/ currentRecord Module	<p>To get the current record type in a client script, use <code>CurrentRecord.type</code>:</p> <pre>function saveRecord(context) { var rec = context.currentRecord; var recType = rec.type; }</pre> <p>To get the current record type in a server-side script, use <code>Record.type</code> in a beforeLoad(scriptContext),</p>

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			<p>beforeSubmit(scriptContext), or afterSubmit(scriptContext) user event script:</p> <pre>function beforeSubmit(context) { var newRec = context.newRecord; var recType = newRec.type; }</pre>
nlapiGetRole()	User.role	N/runtime Module	-
nlapiGetSubsidiary()	User.subsidiary	N/runtime Module	-
nlapiGetUser()	runtime.getCurrentUser()	N/runtime Module	-
nlapiInitiateWorkflow(recordType, id, workflowid, initialValues)	workflow.initiate(options)	N/workflow Module	For a script sample, see N/workflow Module Script Sample.
nlapiInsertLineItem(type, line)	Record.insertLine(options) CurrentRecord.insertLine(options)	N/record Module N/currentRecord Module	-
nlapiInsertLineItemOption(type, fldnam, value, text, selected)	Field.insertSelectOption(options)	N/currentRecord Module	-
nlapiInsertSelectOption(fldnam, value, text, selected)	Field.insertSelectOption(options)	N/currentRecord Module	-
nlapiIsLineItemChanged(type)	Sublist.isChanged	N/record Module	<p>Note that isChanged is a property</p> <pre>record.getSublist("addressbook").isChanged</pre>
nlapiLoadConfiguration(type)	config.load(options)	N/config Module	For a script sample, see N/config Module Script Sample.
nlapiLoadFile(id)	file.load(options)	N/file Module	For a script sample, see N/file Module Script Sample.
nlapiLoadRecord(type, id, initializeValues)	record.load(options)	N/record Module	-
nlapiLoadSearch(type, id)	search.load(options)	N/search Module	For a script sample, see N/search Module Script Samples.
nlapiLogExecution(type, title, details)	log.audit(options) log.debug(options) log.emergency(options) log.error(options)	N/log Module	For a script sample, see log Module Script Sample.
nlapiLookupField(type, id, fields, text)	search.lookupFields(options)	N/search Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiOutboundSSO(id)	sso.generateSuiteSignOnToken(options)		For a script sample, see N/sso Module Script Sample .
nlapiPrintRecord(type, id, mode, properties, incustlocale)	render.bom(options) render.packingSlip(options) render.pickingTicket(options) render.statement(options) render.transaction(options)	N/render Module	For a script sample, see N/render Module Script Sample .
nlapiRefreshLineItems(type)	-	-	This API does not have a SuiteScript 2.0 equivalent.
nlapiRefreshPortlet()	portlet.refresh	N/portlet Module	For a script sample, see N/portlet Module Script Sample
nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)	Record.removeCurrentSublistCurrentRecord.removeCurrentSubrecord(sublist, fldname)	Record.removeCurrentSublistCurrentRecord.removeCurrentSubrecord(sublist, fldname) N/portletSubrecordModule N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiRemoveLineItem(type, line)	Record.removeLine(options) CurrentRecord.removeLine(options)	N/record Module N/currentRecord Module	-
nlapiRemoveLineItemOption(type, fldname, value)	Field.removeSelectOption(options)	N/currentRecord Module	-
nlapiRemoveSelectOption(fldname, value)	Field.removeSelectOption(options)	N/currentRecord Module	-
nlapiRemoveSubrecord(fldname)	Record.removeSubrecord(options) CurrentRecord.removeSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlapiRequestURL(url, postdata, headers, callback, httpMethod)	http.delete(options) http.get(options) http.post(options) http.put(options) http.request(options)	N/http Module	-
nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)	-	https	-
nlapiResizePortlet()	portlet.resize	N/portlet Module	For a script sample, see N/portlet Module Script Sample .
nlapiResolveURL(type, identifier, id, displayMode)	url.resolveRecord(options) url.resolveScript(options) url.resolveTaskLink(options)	N/url Module	For a script sample, see N/url Module Script Samples .
nlapiScheduleScript(scriptId, deployId, params)	task.create(options)	N/task Module	<pre>var scheduleScriptTaskObj = task.create({</pre>

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			<pre> taskType: taskType.SCHEDULED_SCRIPT, // the params }); </pre>
nlapiSearchDuplicate(type, fields, id)	search.duplicates(options)	N/search Module	-
nlapiSearchGlobal(keywords)	search.global(options)	N/search Module	-
nlapiSearchRecord(type, id, filters, columns)	search.create(options) search.load(options)	N/search Module	For a script sample, see N/search Module Script Samples .
nlapiSelectLineItem(type, linenum)	Record.selectLine(options) CurrentRecord.selectLine(options)	N/record Module N/currentRecord Module	-
nlapiSelectNewLineItem(type)	Record.selectNewLine(options) CurrentRecord.selectNewLine(options)	N/record Module N/currentRecord Module	-
nlapiSelectNode(node, xpath)	XPath.select(options)	N/xml Module	-
nlapiSelectNodes(node, xpath)	XPath.select(options)	N/xml Module	-
nlapiSelectValue(node, xpath)	See Notes	N/xml Module	To mimic this functionality in SuiteScript 2.0, select a node with XPath.select(options) and then inspect the Node.textContent property.
nlapiSelectValues(node, path)	See Notes	N/xml Module	To mimic this functionality in SuiteScript 2.0, select an array of nodes with XPath.select(options) and then loop through each node's Node.textContent property.
nlapiSendCampaignEmail(campaigneventid, recipientid)	email.sendCampaignEvent(options)	N/email Module	-
nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo)	email.send(options) email.sendBulk(options)	N/email Module	For a script sample, see N/email Module Script Sample .
nlapiSendFax(author, recipient, subject, body, records, attachments)	N/A	-	This API does not have a SuiteScript 2.0 equivalent.
nlapiSetCurrentLineItemDateValue(type, fieldId, dateTime, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
nlapiSetCurrentLineItemMatrixValue(type, fldname, col)	Record.setCurrentMatrixSublineItemValue(options) CurrentRecord.setCurrentMatrixSublineItemValue(options)	N/record Module N/currentRecord Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
um, value, firefieldchanged, synchronous)		N/ currentRecord Module	
nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)	Record.setCurrentSublistText(options) CurrentRecord.setCurrentSublistText(options)	N/ currentRecord Module	-
nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)	Record.setCurrentSublistValue(options) CurrentRecord.setCurrentSublistValue(options)	N/ currentRecord Module	-
nlapiSetCurrentLineItemValues(type, fldnam, values, firefieldchanged, synchronous)	Record.setCurrentSublistValues(options) CurrentRecord.setCurrentSublistValues(options)	N/ currentRecord Module	-
nlapiSetDateTimeValue(fieldId, dateTime, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)	Record.setText(options) CurrentRecord.setText(options)	N/record Module N/ currentRecord Module	-
nlapiSetFieldTexts(fldname, txts, firefieldchanged, synchronous)	Record.setText(options) CurrentRecord.setText(options)	N/record Module N/ currentRecord Module	-
nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)	Record.setValue(options) CurrentRecord.setValue(options)	N/record Module N/ currentRecord Module	-
nlapiSetFieldValues(fldnam, value, firefieldchanged, synchronous)	Record.setValue(options) CurrentRecord.setValue(options)	N/record Module N/ currentRecord Module	-
nlapiSetLineItemDateTimeValue(type, fieldId, lineNumber, dateTime, timeZone)	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
nlapiSetLineItemValue(type, fldnam, linenumber, value)	Record.setSublistValue(options) CurrentRecord.setSublistValue(options)	N/record Module	-
nlapiSetMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)	Record.setMatrixHeaderValue(options) CurrentRecord.setMatrixHeaderValue(options)	N/ currentRecord Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiSetRecoveryPoint()	See Notes	See Notes	The SuiteScript 2.0 Map/Reduce Script Type automatically incorporates yielding.
nlapiSetRedirectURL(type, identifier, id, editmode, parameters)	redirect.redirect(options) redirect.toRecord(options) redirect.toSuitelet(options) redirect.toTaskLink(options)	N/redirect Module	For a script sample, see N/redirect Module Script Sample.
nlapiStringToDate(str, format)	format.parse(options)	N/format Module	For a script sample, see N/format Module Script Samples.
nlapiStringToXML(text)	Parser.fromString(options)	N/xml Module	-
nlapiSubmitConfiguration(name)	Record.save(options)	N/record Module	-
nlapiSubmitCSVImport(nlobjCSVImport)		N/task Module	-
nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)	Record.save(options)	N/record Module	-
nlapiSubmitField(type, id, fields, values, doSourcing)	record.submitFields(options)	N/record Module	-
nlapiSubmitFile(file)	File.save()	N/file Module	For a script sample, see N/file Module Script Sample.
nlapiTransformRecord(type, id, transformType, transformValues)	record.transform(options)	N/record Module	-
nlapiTriggerWorkflow(recordtype, id, workflowid, actionid, stateid)	workflow.trigger(options)	N/workflow Module	-
nlapiValidateXML(xmlDocument, schemaDocument, schemaFolderId)	xml.validate(options)	N/xml Module	-
nlapiViewCurrentLineItemSubrecord(sublist, fldname)	CurrentRecord.getCurrentSublistSubrecord(options) Record.getCurrentSublistSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords.
nlapiViewLineItemSubrecord(sublist, fldname, linenumber)	Record.getSublistSubrecord(options)	N/record Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords.
nlapiViewSubrecord(fldname)	Record.getSubrecord(options) CurrentRecord.getSubrecord(options)	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords.
nlapiVoidTransaction(transactionType, recordId)	transaction.void(options)	N/transaction Module	For a script sample, see N/transaction Module Script Sample.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlapiXMLToPDF(xmlstring)	render.xmlToPdf(options) TemplateRenderer.renderAsPdf()	N/render Module	Note that <code>TemplateRenderer.renderAsPdf()</code> is equivalent to <code>nlapiXMLToPDF(nlobjEmailMerger.renderToString())</code> . For a script sample, see N/render Module Script Sample .
nlapiXMLToString(xml)	Parser.toString(options)	N/xml Module	-
nlapiYieldScript()	See Notes	See Notes	Note that the SuiteScript 2.0 Map/Reduce Script Type automatically incorporates yielding.

SuiteScript 1.0 to SuiteScript 2.0 API Map – Objects (nlobj)

This topic maps SuiteScript 1.0 Objects (prefixed with “nlobj”) to their corresponding SuiteScript 2.0 APIs. Objects and their members are grouped alphabetically by object name. Each object has its own table containing all object members.

Note: To view a mapping of SuiteScript 1.0 Functions (prefixed with “nlapi”) to their corresponding SuiteScript 2.0 APIs, see [SuiteScript 1.0 to SuiteScript 2.0 API Map – Functions \(nlapi\)](#).

- nlobjAssistant
- nlobjButton
- nlobjButton
- nlobjColumn
- nlobjConfiguration
- nlobjContext
- nlobjCredentialBuilder
- nlobjCSVImport
- nlobjDuplicateJobRequest
- nlobjEmailMerger
- nlobjError
- nlobjField
- nlobjFieldGroup
- nlobjFile
- nlobjForm
- nlobjFuture
- nlobjJobManager
- nlobjList
- nlobjLogin
- nlobjMergeResult
- nlobjPortlet
- nlobjRecord
- nlobjRequest

- nlobjResponse
- nlobjSearch
- nlobjSearchColumn
- nlobjSearchFilter
- nlobjSearchResult
- nlobjSearchResultSet
- nlobjSelectOption
- nlobjSublist
- nlobjSubrecord
- nlobjTab
- nlobjTemplateRenderer

nlobjAssistant

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjAssistant	serverWidget.Assistant	N/ui/serverWidget Module	-
nlobjAssistant.addField(name, type, label, source, group)	Assistant.addField(options)	N/ui/serverWidget Module	-
nlobjAssistant.addFieldGroup(name, label)	Assistant.addFieldGroup(options)	N/ui/serverWidget Module	-
nlobjAssistant.addStep(name, label)	Assistant.addStep(options)	N/ui/serverWidget Module	-
nlobjAssistant.addSubList(name, type, label)	Assistant.addSublist(options)	N/ui/serverWidget Module	-
nlobjAssistant.getAllFields()	Assistant.getFieldIds()	N/ui/serverWidget Module	-
nlobjAssistant.getAllFieldGroups()	Assistant.getFieldGroupIds()	N/ui/serverWidget Module	-
nlobjAssistant.getAllSteps()	Assistant.getSteps()	N/ui/serverWidget Module	-
nlobjAssistant.getAllSubLists()	Assistant.getSublistIds()	N/ui/serverWidget Module	-
nlobjAssistant.getCurrentStep()	Assistant.currentStep	N/ui/serverWidget Module	Note that <code>currentStep</code> is a property.
nlobjAssistant.getField(name)	Assistant.getField(options)	N/ui/serverWidget Module	-
nlobjAssistant.getFieldGroup(name)	Assistant.getFieldGroup(options)	N/ui/serverWidget Module	-
nlobjAssistant.getLastAction()	Assistant.getLastAction()	N/ui/serverWidget Module	-
nlobjAssistant.getLastStep()	Assistant.getLastStep()	N/ui/serverWidget Module	-
nlobjAssistant.getNextStep()	Assistant.getNextStep()	N/ui/serverWidget Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjAssistant.getStep(name)</code>	<code>Assistant.getStep(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.getSubList(name)</code>	<code>Assistant.getSublist(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.hasError()</code>	<code>Assistant.hasErrorHtml()</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.isFinished()</code>	<code>Assistant.isFinished()</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.sendRedirect(response)</code>	<code>Assistant.sendRedirect(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.setCurrentStep(step)</code>	<code>Assistant.currentStep</code>	<code>N/ui/serverWidget Module</code>	Note that <code>currentStep</code> is a property.
<code>nlobjAssistant.setError(html)</code>	<code>Assistant.errorHtml</code>	<code>N/ui/serverWidget Module</code>	Note that <code>errorHtml</code> is a property.
<code>nlobjAssistant.setFieldValues(values)</code>	<code>Assistant.updateDefaultValues(values)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.setFinished(html)</code>	<code>Assistant.finishedHtml</code>	<code>N/ui/serverWidget Module</code>	Note that <code>finishedHtml</code> is a property.
<code>nlobjAssistant.setNumbered(hasStepNumber)</code>	<code>Assistant.hideStepNumber</code>	<code>N/ui/serverWidget Module</code>	Note that <code>hideStepNumber</code> is a property.
<code>nlobjAssistant.setOrdered(ordered)</code>	<code>Assistant.isNotOrdered</code>	<code>N/ui/serverWidget Module</code>	Note that <code>isNotOrdered</code> is a property.
<code>nlobjAssistant.setScript(script)</code>	<code>Assistant.clientScriptFileId</code> <code>Assistant.clientScriptModulePath</code>	<code>N/ui/serverWidget Module</code>	Note that <code>clientScriptFileId</code> and <code>clientScriptModulePath</code> are properties. Use one of these SuiteScript 2.0 properties to attach an ad hoc client script to an assistant.
<code>nlobjAssistant.setShortcut(show)</code>	<code>Assistant.hideAddToShortcutsLink</code>	<code>N/ui/serverWidget Module</code>	Note that <code>hideAddToShortcutsLink</code> is a property.
<code>nlobjAssistant.setSplash(title, text1, text2)</code>	<code>Assistant.setSplash(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistant.setTitle(title)</code>	<code>Assistant.title</code>	<code>N/ui/serverWidget Module</code>	Note that <code>title</code> is a property.

nlobjAssistantStep

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjAssistantStep</code>	<code>serverWidget.AssistantStep</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistantStep.getAllFields()</code>	<code>AssistantStep.getFieldIds()</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjAssistantStep.getAllLineItemFieldsAsGroups()</code>	<code>AssistantStep.getSublistFieldIds(options)</code>	<code>N/ui/serverWidget Module</code>	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjAssistantStep.getAllLineItems()	AssistantStep.getSubmittedSublistIds()	N/ui/serverWidget Module	-
nlobjAssistantStep.getFieldValue(name)	AssistantStep.getValue(options)	N/ui/serverWidget Module	-
nlobjAssistantStep.getFieldValues(name)	AssistantStep.getValue(options)	N/ui/serverWidget Module	-
nlobjAssistantStep.getLineItemCount()	AssistantStep.getLineCount(options)	N/ui/serverWidget Module	-
nlobjAssistantStep.getLineItemValue(assistantStepName, line)	AssistantStep.getSublistValue(options)	N/ui/serverWidget Module	-
nlobjAssistantStep.getStepNumber()	AssistantStep.stepNumber	N/ui/serverWidget Module	Note that stepNumber is a property.
nlobjAssistantStep.setHelpText(help)	AssistantStep.helpText	N/ui/serverWidget Module	Note that helpText is a property.
nlobjAssistantStep.setLabel(label)	AssistantStep.label	N/ui/serverWidget Module	Note that label is a property.

nlobjButton

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjButton	serverWidget.Button	N/ui/serverWidget Module	-
nlobjButton.setDisabled(disabled)	Button.isDisabled	N/ui/serverWidget Module	Note that isDisabled is a property.
nlobjButton.setLabel(label)	Button.label	N/ui/serverWidget Module	Note that label is a property.
nlobjButton.setVisible(visible)	Button.isHidden	N/ui/serverWidget Module	Note that isHidden is a property.

nlobjColumn

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjColumn	serverWidget.ListColumn	N/ui/serverWidget Module	-
nlobjColumn.addParamToURL(param, value, dynamic)	ListColumn.addParamToURL(options)	N/ui/serverWidget Module	-
nlobjColumn.setLabel(label)	ListColumn.label	N/ui/serverWidget Module	Note that label is a property.
nlobjColumn.setURL(url, dynamic)	ListColumn.setURL(options)	N/ui/serverWidget Module	-

nlobjConfiguration

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjConfiguration	record.Record	N/record Module	Use the N/config Module method, config.load(options) , to return a <code>record.Record</code> object. Then use the <code>record.Record</code> object members to access the specified configuration page. For a script sample, see N/config Module Script Sample .
nlobjConfiguration.getAllFields()	Record.getFields()	N/record Module	-
nlobjConfiguration.getField(fieldName)	Record.getField(options)	N/record Module	-
nlobjConfiguration.getFieldText(fieldName)	Record.getText(options)	N/record Module	-
nlobjConfiguration.getFieldTextByRecordId(recordId)	Record.getText(options)	N/record Module	-
nlobjConfiguration.getFieldValue(fieldName)	Record.getValue(options)	N/record Module	-
nlobjConfiguration.getFieldValueByRecordId(recordId)	Record.getValue(options)	N/record Module	-
nlobjConfiguration.getType()	Record.type	N/record Module	Note that <code>type</code> is a property.
nlobjConfiguration.setFieldText(fieldName, text)	Record.setText(options)	N/record Module	-
nlobjConfiguration.setFieldTextByRecordId(recordId, text)	Record.setText(options)	N/record Module	-
nlobjConfiguration.setFieldValue(fieldName, value)	Record.setValue(options)	N/record Module	-
nlobjConfiguration.setFieldValueByRecordId(recordId, value)	Record.setValue(options)	N/record Module	-

nlobjContext

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjContext	runtime.Script runtime.Session runtime.User	N/runtime Module	-
nlobjContext.getCompany()	runtime.accountId	N/runtime Module	Note that <code>accountId</code> is a property.
nlobjContext.getDepartment()	User.department	N/runtime Module	Note that <code>department</code> is a property.
nlobjContext.getDeploymentId()	Script.deploymentId	N/runtime Module	Note that <code>deploymentId</code> is a property.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjContext.getEmail()</code>	<code>User.email</code>	<code>N/runtime Module</code>	Note that <code>email</code> is a property.
<code>nlobjContext.getEnvironment()</code>	<code>runtime.envType</code>	<code>N/runtime Module</code>	Note that <code>envType</code> is a property.
<code>nlobjContext.getExecutionContext()</code>	<code>runtime.executionContext</code>	<code>N/runtime Module</code>	Note that <code>executionContext</code> is a property.
<code>nlobjContext.getFeature(name)</code>	<code>runtime.isFeatureInEffect(options)</code>	<code>N/runtime Module</code>	-
<code>nlobjContext.getLocation()</code>	<code>User.location</code>	<code>N/runtime Module</code>	Note that <code>location</code> is a property.
<code>nlobjContext.getLogLevel()</code>	<code>Script.logLevel</code>	<code>N/runtime Module</code>	Note that <code>logLevel</code> is a property.
<code>nlobjContext.getName()</code>	<code>User.name</code>	<code>N/runtime Module</code>	Note that <code>name</code> is a property.
<code>nlobjContext.getPercentComplete()</code>	<code>Script.percentComplete</code>	<code>N/runtime Module</code>	Note that <code>percentComplete</code> is a property. For a script sample, see N/runtime Module Script Sample .
<code>nlobjContext.getPermission(name)</code>	<code>User.getPermission(options)</code>	<code>N/runtime Module</code>	-
<code>nlobjContext.getPreference(name)</code>	<code>User.getPreference(options)</code>	<code>N/runtime Module</code>	-
<code>nlobjContext.getQueueCount()</code>	<code>runtime.queueCount</code>	<code>N/runtime Module</code>	Note that <code>queueCount</code> is a property.
<code>nlobjContext.getRemainingUsage()</code>	<code>Script.getRemainingUsage()</code>	<code>N/runtime Module</code>	-
<code>nlobjContext.getRole()</code>	<code>User.role</code>	<code>N/runtime Module</code>	Note that <code>role</code> is a property.
<code>nlobjContext.getRoleCenter()</code>	<code>User.roleCenter</code>	<code>N/runtime Module</code>	Note that <code>roleCenter</code> is a property.
<code>nlobjContext.getRoleId()</code>	<code>User.roleId</code>	<code>N/runtime Module</code>	Note that <code>roleId</code> is a property.
<code>nlobjContext.getScriptId()</code>	<code>Script.id</code>	<code>N/runtime Module</code>	Note that <code>id</code> is a property.
<code>nlobjContext.getSessionObject(name)</code>	<code>Session.get(options)</code>	<code>N/runtime Module</code>	-
<code>nlobjContext.getSetting(type, name)</code>	<code>Script.getParameter(options)</code> <code>Session.get(options)</code> <code>runtime.isFeatureInEffect(options)</code> <code>User.getPermission(options)</code>	<code>N/runtime Module</code>	The method <code>Script.getParameter(options)</code> is equivalent to <code>nlobjContext.getSetting('SCRIPT', name)</code> . The method <code>Session.get(options)</code> is equivalent to <code>nlobjContext.getSetting('SESSION', name)</code> . The method <code>runtime.isFeatureInEffect(options)</code> is equivalent to <code>nlobjContext.getSetting('FEATURE', name)</code> .

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			The method <code>User.getPermission(options)</code> is equivalent to <code>nlobjContext.getSetting('PERMISSION', name)</code>
<code>nlobjContext.getSubsidiary()</code>	<code>User.subsidiary</code>	N/runtime Module	Note that <code>subsidiary</code> is a property.
<code>nlobjContext.getUser()</code>	<code>User.id</code>	N/runtime Module	Note that <code>id</code> is a property.
<code>nlobjContext.getVersion()</code>	<code>runtime.version</code>	N/runtime Module	Note that <code>version</code> is a property.
<code>nlobjContext.setPercentComplete(pct)</code>	<code>Script.percentComplete</code>	N/runtime Module	Note that <code>percentComplete</code> is a property.
<code>nlobjContext.setSessionObject(name, value)</code>	<code>Session.set(options)</code>	N/runtime Module	-
<code>nlobjContext.setSetting(type, name, value)</code>	<code>Session.set(options)</code>	N/runtime Module	-

nlobjCredentialBuilder

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjCredentialBuilder(string, domainString)</code>	<code>https.SecureString</code>	N/https Module	-
<code>nlobjCredentialBuilder.append(nlobjCredentialBuilder, string)</code>	<code>SecureString.appendSecureString(options)</code> <code>SecureString.appendString(options)</code>	N/https Module	-
<code>nlobjCredentialBuilder.base64()</code>	<code>SecureString.convertEncoding(options)</code>	N/https Module	-
<code>nlobjCredentialBuilder.md5()</code>	<code>SecureString.hash(options)</code>	N/https Module	-
<code>nlobjCredentialBuilder.replace(string1, string2)</code>		N/https Module	-
<code>nlobjCredentialBuilder.sha1()</code>	<code>SecureString.hash(options)</code>	N/https Module	-
<code>nlobjCredentialBuilder.sha256()</code>	<code>SecureString.hash(options)</code>	N/https Module	-
<code>nlobjCredentialBuilder.utf8()</code>	<code>SecureString.convertEncoding(options)</code>	N/https Module	-

nlobjCSVImport

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjCSVImport</code>	<code>task.CsvImportTask</code>	N/task Module	Returned by <code>task.create(options)</code> . <pre>var csvImpTaskObj = task.create({</pre>

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			<pre>taskType: task.TaskType.CSV_IMPORT, //Other Params });</pre>
nlobjCSVImport.setLinkedFile(sublist, file)	CsvImportTask.linkedFiles	N/task Module	Note that linkedFiles is a property.
nlobjCSVImport.setMapping(savedImport, mappingId)	CsvImportTask.mappingId	N/task Module	Note that mappingId is a property.
nlobjCSVImport.setOption(option, value)	CsvImportTask.name	N/task Module	Note that name is a property.
nlobjCSVImport.setPrimaryFile(file)	CsvImportTask.importFile	N/task Module	Note that importFile is a property.
nlobjCSVImport.setQueue(string)	CsvImportTask.queueId	N/task Module	Note that queueId is a property.

nlobjDuplicateJobRequest

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjDuplicateJobRequest	task.EntityDeduplicationTask	N/task Module	<p>Returned by <code>task.create(options)</code>.</p> <pre>var dedupTaskObj = task.create({ taskType: task.TaskType.ENTITY_DEDUPLICATION, //Other Params });</pre>
nlobjDuplicateJobRequest.setEntityType(entityType)	EntityDeduplicationTask.entityType	N/task Module	Note that entityType is a property.
nlobjDuplicateJobRequest.setMasterId(masterID)	EntityDeduplicationTask.masterRecordId	N/task Module	-
nlobjDuplicateJobRequest.setMasterSelectionMode(mode)	EntityDeduplicationTask.masterSelectionMode	N/task Module	Note that masterSelectionMode is a property.
nlobjDuplicateJobRequest.setOperation(operation)	EntityDeduplicationTask.dedupeMode	N/task Module	Note that dedupeMode is a property.
nlobjDuplicateJobRequest.setRecords(dupeRecords)	EntityDeduplicationTask.recordIds	N/task Module	Note that recordIds is a property.

nlobjEmailMerger

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjEmailMerger	render.EmailMergeResult	N/render Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjEmailMerger.merge()</code>		<code>N/render Module</code>	-
<code>nlobjEmailMerger.setCustomRecord(recordType, recordId)</code>	See Notes	<code>N/render Module</code>	In SuiteScript 2.0, this value is set with a <code>render.mergeEmail(options)</code> parameter.
<code>nlobjEmailMerger.setEntity(entityType, entityId)</code>	See Notes	<code>N/render Module</code>	In SuiteScript 2.0, this value is set with a <code>render.mergeEmail(options)</code> parameter.
<code>nlobjEmailMerger.setRecipient(recipientType, recipientId)</code>	See Notes	<code>N/render Module</code>	In SuiteScript 2.0, this value is set with a <code>render.mergeEmail(options)</code> parameter.
<code>nlobjEmailMerger.setSupportCase(caseId)</code>	See Notes	<code>N/render Module</code>	In SuiteScript 2.0, this value is set with a <code>render.mergeEmail(options)</code> parameter.
<code>nlobjEmailMerger.setTransaction(transactionId)</code>	See Notes	<code>N/render Module</code>	In SuiteScript 2.0, this value is set with a <code>render.mergeEmail(options)</code> parameter.

nlobjError

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjError</code>	<code>error.SuiteScriptError</code> <code>error.UserEventError</code>	<code>N/error Module</code>	-
<code>nlobjError.getCode()</code>	-	<code>N/error Module</code>	-
<code>nlobjError.getDetails()</code>	-	<code>N/error Module</code>	-
<code>nlobjError.getId()</code>	-	<code>N/error Module</code>	-
<code>nlobjError.getInternalId()</code>	-	<code>N/error Module</code>	-
<code>nlobjError.getStackTrace()</code>	-	<code>N/error Module</code>	-
<code>nlobjError.getUserEvent()</code>	-	<code>N/error Module</code>	-

nlobjField

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjField</code>	<code>serverWidget.Field</code> <code>record.Field</code>	<code>N/ui/serverWidget Module</code> <code>N/record Module</code>	Use the <code>N/ui/serverWidget</code> module to create and modify form fields in a Suitelet. Use the <code>N/record</code> module to access field metadata in client and server-side scripts.

nlobjFieldGroup

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjFieldGroup	serverWidget.FieldGroup	N/ui/serverWidget Module	-
nlobjFieldGroup.setCollapsible(collapseHidden)	FieldGroup.isCollapsible	N/ui/serverWidget Module	Note that <code>isCollapsible</code> is a property.
nlobjFieldGroup.setLabel(label)	FieldGroup.label	N/ui/serverWidget Module	Note that <code>label</code> is a property.
nlobjFieldGroup.setShowBorder(showBorder)	FieldGroup.isBorderHidden	N/ui/serverWidget Module	Note that <code>isBorderHidden</code> is a property.
nlobjFieldGroup.setSingleColumn(column)	FieldGroup.isSingleColumn	N/ui/serverWidget Module	Note that <code>isSingleColumn</code> is a property.

nlobjFile

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjFile	file.File	N/file Module	For a script sample, see N/file Module Script Sample .
nlobjFile.getDescription()	File.description	N/file Module	Note that <code>description</code> is a property.
nlobjFile.getFolder()	File.folder	N/file Module	Note that <code>folder</code> is a property. For a script sample, see N/file Module Script Sample .
nlobjFile.getId()	File.id	N/file Module	Note that <code>id</code> is a property. For a script sample, see N/file Module Script Sample .
nlobjFile.getName()	File.name	N/file Module	Note that <code>name</code> is a property. For a script sample, see N/file Module Script Sample .
nlobjFile.getSize()	File.size	N/file Module	Note that <code>size</code> is a property.
nlobjFile.getType()	File.fileType	N/file Module	Note that <code>fileType</code> is a property. For a script sample, see N/file Module Script Sample . For a script sample, see N/file Module Script Sample .
nlobjFile.getURL()	File.url	N/file Module	Note that <code>url</code> is a property.
nlobjFile.getValue()	File.getContents()	N/file Module	-
nlobjFile.isInactive()	File.isInactive	N/file Module	Note that <code>isInactive</code> is a property.
nlobjFile.isOnline()	File.isOnline	N/file Module	Note that <code>isOnline</code> is a property. For a script sample, see N/file Module Script Sample .

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjFile.setDescription(description)</code>	<code>File.description</code>	<code>N/file Module</code>	Note that <code>description</code> is a property.
<code>nlobjFile.setEncoding(encodingType)</code>	<code>File.encoding</code>	<code>N/file Module</code>	Note that <code>encoding</code> is a property.
<code>nlobjFile.setFolder(id)</code>	<code>File.folder</code>	<code>N/file Module</code>	Note that <code>folder</code> is a property. You can also set the folder during file creation with <code>file.create(options)</code> . For a script sample, see N/file Module Script Sample .
<code>nlobjFile.setIsInactive(inactive)</code>	<code>File.isInactive</code>	<code>N/file Module</code>	Note that <code>isInactive</code> is a property.
<code>nlobjFile.setIsOnline(online)</code>	<code>File.isOnline</code>	<code>N/file Module</code>	Note that <code>isOnline</code> is a property. For a script sample, see N/file Module Script Sample .
<code>nlobjFile.setName(name)</code>	<code>File.name</code>	<code>N/file Module</code>	Note that <code>name</code> is a property. For a script sample, see N/file Module Script Sample .

nlobjForm

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjForm</code>	<code>serverWidget.Form</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addButton(name, label, script)</code>	<code>Form.addButton(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addCredentialField(id, label, website, scriptId, value, entityMatch, tab)</code>	<code>Form.addCredentialField(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addField(name, type, label, sourceOrRadio, tab)</code>	<code>Form.addField(options)</code>	<code>N/ui/serverWidget Module</code>	For a script sample, see N/ui/serverWidget Module Script Samples
<code>nlobjForm.addFieldGroup(name, label, tab)</code>	<code>Form.addFieldGroup(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addPageLink(type, title, url)</code>	<code>Form.addPageLink(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addResetButton(label)</code>	<code>Form.addResetButton(options)</code>	<code>N/ui/serverWidget Module</code>	-
<code>nlobjForm.addSubList(name, type, label, tab)</code>	<code>Form.addSublist(options)</code>	<code>N/ui/serverWidget Module</code>	For a script sample, see N/ui/serverWidget Module Script Samples
<code>nlobjForm.addSubmitButton(label)</code>	<code>Form.addSubmitButton(options)</code>	<code>N/ui/serverWidget Module</code>	For a script sample, see N/ui/serverWidget Module Script Samples
<code>nlobjForm.addSubTab(name, label, tab)</code>	<code>Form.addSubtab(options)</code>	<code>N/ui/serverWidget Module</code>	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjForm.addTab(name, label)	Form.addTab(options)	N/ui/serverWidget Module	-
nlobjForm.getButton(name)	Form.getButton(options)	N/ui/serverWidget Module	-
nlobjForm.getField(name, radio)	Form.getField(options)	N/ui/serverWidget Module	-
nlobjForm.getSubList(name)	Form.getSublist(options)	N/ui/serverWidget Module	-
nlobjForm.getSubTab(name)	Form.getSubtab(options)	N/ui/serverWidget Module	-
nlobjForm.getTab(name)	Form.getTab(options)	N/ui/serverWidget Module	-
nlobjForm.getTabs()	Form.getTabs()	N/ui/serverWidget Module	-
nlobjForm.insertField(field, nextfld)	Form.insertField(options)	N/ui/serverWidget Module	-
nlobjForm.insertSubList(sublist, nextsub)	Form.insertSublist(options)	N/ui/serverWidget Module	-
nlobjForm.insertSubTab(subtab, nextsub)	Form.insertSubtab(options)	N/ui/serverWidget Module	-
nlobjForm.insertTab(tab, nexttab)	Form.insertTab(options)	N/ui/serverWidget Module	-
nlobjForm.removeButton(name)	Form.removeButton(options)	N/ui/serverWidget Module	-
nlobjForm.setFieldValues(values)	Form.updateDefaultValues(options)	N/ui/serverWidget Module	-
nlobjForm.setScript(script)	Form.clientScriptFileId Form.clientScriptModulePath	N/ui/serverWidget Module	Note that <code>clientScriptFileId</code> and <code>clientScriptModulePath</code> are properties. Use one of these SuiteScript 2.0 properties to attach an ad hoc client script to a form.
nlobjForm.setTitle(title)	Form.title	N/ui/serverWidget Module	Note that <code>title</code> is a property.

nlobjFuture

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjFuture	task.EntityDeduplicationTaskStatus	N/task Module	-
nlobjFuture.isCancelled()	EntityDeduplicationTaskStatus.status	N/task Module	Note that <code>status</code> is a property.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjFuture.isDone()	EntityDeduplicationTaskStatus.status	N/task Module	Note that status is a property.

nlobjJobManager

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjJobManager	-	N/task Module	-
nlobjJobManager.createJobRequest()	-	N/task Module	-
nlobjJobManager.getFuture()	-	N/task Module	-
nlobjJobManager.submit(nlobjDuplicateJobRequest)	-	N/task Module	-

nlobjList

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjList	serverWidget.List	N/ui/serverWidget Module	-
nlobjList.addButton(name, label, script)	List.addButton(options)	N/ui/serverWidget Module	-
nlobjList.addColumn(name, type, label, align)	List.addColumn(options)	N/ui/serverWidget Module	-
nlobjList.addEditColumn(column, showView, showHrefCol)	List.addEditColumn(options)	N/ui/serverWidget Module	-
nlobjList.addPageLink(type, title, url)	List.addPageLink(options)	N/ui/serverWidget Module	-
nlobjList.addRow(row)	List.addRow(options)	N/ui/serverWidget Module	-
nlobjList.addRows(rows)	List.addRows(options)	N/ui/serverWidget Module	-
nlobjList.setScript(script)	List.clientScriptFileId List.clientScriptModulePath	N/ui/serverWidget Module	Note that clientScriptFileId and clientScriptModulePath are properties. Use one of these SuiteScript 2.0 properties to attach an ad hoc client script to a form.
nlobjList.setStyle(style)	List.style	N/ui/serverWidget Module	Note that style is a property.
nlobjList.setTitle(title)	List.title	N/ui/serverWidget Module	Note that title is a property.

nlobjLogin

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjLogin	See Notes.	N/auth Module	See nlobjLogin members.
nlobjLogin.changeEmail(currentPassword, newEmail, justThisAccount)	auth.changeEmail(options)	N/auth Module	For a script sample, see N/auth Module Script Sample .
nlobjLogin.changePassword(currentPassword, newPassword)	auth.changePassword(options)	N/auth Module	For a script sample, see N/auth Module Script Sample .

nlobjMergeResult

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjMergeResult	render.EmailMergeResult	N/render Module	-
nlobjMergeResult.getBody()	EmailMergeResult.body	N/render Module	Note that body is a property.
nlobjMergeResult.getSubject()	EmailMergeResult.subject	N/render Module	Note that subject is a property.

nlobjPortlet

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjPortlet	Portlet Object	See Notes	For additional information, see SuiteScript 2.0 Portlet Script Type .
nlobjPortlet.addColumn(name, type, label, just)	Portlet.addColumn(options)	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
nlobjPortlet.addEditColumn(column, showView, showHrefCol)	Portlet.addEditColumn(options)	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
nlobjPortlet.addField(name, type, label, source)	Portlet.addField(options)	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
nlobjPortlet.addLine(text, url, indent)	Portlet.addLine(options)	-	For additional information, see SuiteScript 2.0 Portlet Script Type .

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjPortlet.addRow(row)</code>	<code>Portlet.addRow(options)</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.addRows(rows)</code>	<code>Portlet.addRows(options)</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.setHtml(html)</code>	<code>Portlet.html</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.setRefreshInterval(n)</code>		-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.setScript(scriptid)</code>	<code>Portlet.clientScriptFileId</code> <code>Portlet.clientScriptModulePath</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.setSubmitButton(url, label, target)</code>	<code>Portlet.setSubmitButton(options)</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .
<code>nlobjPortlet.setTitle(title)</code>	<code>Portlet.title</code>	-	For additional information, see SuiteScript 2.0 Portlet Script Type .

nlobjRecord

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjRecord</code>	<code>record.Record</code> <code>currentRecord.CurrentRecord</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.commitLineItem(group, ignoreRecalc)</code>	<code>Record.commitLine(options)</code> <code>CurrentRecord.commitLine(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.createCurrentLineItemSubrecord(sublist, fldname)</code>	<code>Record.getCurrentSublistSubrecord(options)</code> <code>CurrentRecord.getCurrentSublistSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			2.0 Scripting Subrecords.
<code>nlobjRecord.createSubrecord(fldname)</code>	<code>Record.getSubrecord(options)</code> <code>CurrentRecord.getSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.editCurrentLineItemSubrecord(sublist, fldname)</code>	<code>Record.getCurrentSublistSubrecord(options)</code> <code>CurrentRecord.getCurrentSublistSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.editSubrecord(fldname)</code>	<code>Record.getSubrecord(options)</code> <code>CurrentRecord.getSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.findLineItemMatrixValue(group, fldnam, column, val)</code>	<code>Record.findMatrixSublistLineWithValue(options)</code> <code>CurrentRecord.findMatrixSublistLineWithValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.findLineItemValue(group, fldnam, value)</code>	<code>Record.findSublistLineWithValue(options)</code> <code>CurrentRecord.findSublistLineWithValue(options)</code>	N/record Module N/currentRecord Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjRecord.getAllFields()</code>	<code>Record.getFields()</code>	N/record Module	-
<code>nlobjRecord.getAllLineItemFields(group)</code>	<code>Record.getSublistFields(options)</code>	N/record Module	-
<code>nlobjRecord.getCurrentLineItemDateTimeValue(type, fieldId, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.getCurrentLineItemMatrixValue(group, fldnam, column)</code>	<code>Record.getCurrentMatrixSublistValue(options)</code> <code>CurrentRecord.getCurrentMatrixSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getCurrentLineItemValue(type, fldnam)</code>	<code>Record.getCurrentSublistValue(options)</code> <code>CurrentRecord.getCurrentSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getCurrentLineItemValues(type, fldnam)</code>	<code>Record.getCurrentSublistValue(options)</code> <code>CurrentRecord.getCurrentSublistValue(options)</code>	N/record Module N/currentRecord Module	Method returns an array for multi-select fields.
<code>nlobjRecord.getDateTimeValue(fieldId, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.getField(fldnam)</code>	<code>Record.getField(options)</code> <code>CurrentRecord.getField(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getFieldText(name)</code>	<code>Record.getText(options)</code> <code>CurrentRecord.getText(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getFieldTexts(name)</code>	<code>Record.getText(options)</code> <code>CurrentRecord.getText(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getFieldValue(name)</code>	<code>Record.getValue(options)</code> <code>CurrentRecord.getValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getFieldValues(name)</code>	<code>Record.getValue(options)</code> <code>CurrentRecord.getValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getId()</code>	<code>Record.id</code>	N/record Module	-
<code>nlobjRecord.getLineItemCount(group)</code>	<code>Record.getLineCount(options)</code>	N/record Module	-
<code>nlobjRecord.getLineItemDateTimeValue(type, fieldId, lineNum, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.getLineItemField(group, fldnam, linenum)</code>	<code>Record.getSublistField(options)</code> <code>CurrentRecord.getSublistField(options)</code>	N/record Module N/currentRecord Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjRecord.getLineItemMatrixField(group, fldnam, linenum, column)</code>	<code>Record.getMatrixSublistField(options)</code> <code>CurrentRecord.getMatrixSublistField(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getLineItemMatrixValue(group, fldnam, linenum, column)</code>	<code>Record.getMatrixSublistValue(options)</code> <code>CurrentRecord.getMatrixSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getLineItemText(group, fldnam, linenum)</code>	<code>Record.getSublistText(options)</code> <code>CurrentRecord.getSublistText(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getLineItemValue(group, name, linenum)</code>	<code>Record.getSublistValue(options)</code> <code>CurrentRecord.getSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getLineItemValues(type, fldnam, linenum)</code>	<code>Record.getSublistValue(options)</code> <code>CurrentRecord.getSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getMatrixCount(group, fldnam)</code>	<code>Record.getMatrixHeaderCount(options)</code> <code>CurrentRecord.getMatrixHeaderCount(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getMatrixField(group, fldname, column)</code>	<code>Record.getMatrixHeaderField(options)</code> <code>CurrentRecord.getMatrixHeaderField(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getMatrixValue(group, fldnam, column)</code>	<code>Record.getMatrixHeaderValue(options)</code> <code>CurrentRecord.getMatrixHeaderValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.getRecordType()</code>	<code>Record.type</code>	N/record Module	-
<code>nlobjRecord.insertLineItem(group, linenum, ignoreRecalc)</code>	<code>Record.insertLine(options)</code> <code>CurrentRecord.insertLine(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.removeLineItem(group, linenum, ignoreRecalc)</code>	<code>Record.removeLine(options)</code> <code>CurrentRecord.removeLine(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.removeCurrentLineItemSubrecord(sublist, fldname)</code>	<code>Record.removeCurrentSublistSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.removeSubrecord(fldname)</code>	<code>Record.removeSubrecord(options)</code> <code>CurrentRecord.removeSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally dif

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			ferent from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords.
<code>nlobjRecord.selectLineItem(group, linenum)</code>	<code>Record.selectLine(options)</code> <code>CurrentRecord.selectLine(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.selectNewLineItem(group)</code>	<code>Record.selectNewLine(options)</code> <code>CurrentRecord.selectNewLine(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.setCurrentLineItemMatrixValue(group, fldnam, column, value)</code>	<code>Record.setCurrentMatrixSublistValue(options)</code> <code>CurrentRecord.setCurrentMatrixSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setCurrentLineItemValue(group, name, value)</code>	<code>Record.setCurrentSublistValue(options)</code> <code>CurrentRecord.setCurrentSublistValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setDateTimeValue(fieldId, dateTime, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.setFieldText(name, text)</code>	<code>Record.setText(options)</code> <code>CurrentRecord.setText(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setFieldTexts(name, text)</code>	<code>Record.setText(options)</code> <code>CurrentRecord.setText(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setFieldValue(name, value)</code>	<code>Record.setValue(options)</code> <code>CurrentRecord.setValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setFieldValues(name, value)</code>	<code>Record.setValue(options)</code> <code>CurrentRecord.setValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.setLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)</code>	See Notes	N/format Module	Use the N/format module to mimic this functionality in SuiteScript 2.0.
<code>nlobjRecord.setLineItemValue(group, name, linenum, value)</code>	<code>Record.setSublistValue(options)</code>	N/record Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
<code>nlobjRecord.setMatrixValue(group, fldnam, column, value)</code>	<code>Record.setMatrixHeaderValue(options)</code> <code>CurrentRecord.setMatrixHeaderValue(options)</code>	N/record Module N/currentRecord Module	-
<code>nlobjRecord.viewCurrentLineItemSubrecord(sublist, fldname)</code>	<code>Record.getCurrentSublistSubrecord(options)</code> <code>CurrentRecord.getCurrentSublistSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.viewLineItemSubrecord(sublist, fldname, linenumber)</code>	<code>Record.getSublistSubrecord(options)</code>	N/record Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
<code>nlobjRecord.viewSubrecord(fldname)</code>	<code>Record.getSubrecord(options)</code> <code>CurrentRecord.getSubrecord(options)</code>	N/record Module N/currentRecord Module	Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .

nlobjRequest

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjRequest	http.ServerRequest	-	-
nlobjRequest.getAllHeaders()	-	-	-
nlobjRequest.getAllParameters()	-	-	-
nlobjRequest.getBody()	-	-	-
nlobjRequest.getFile(id)	-	-	-
nlobjRequest.getHeader(name)	-	-	-
nlobjRequest.getLineItemCount(group)	-	-	-
nlobjRequest.getLineItemValue(group, name, line)	-	-	-
nlobjRequest.getMethod()	-	-	-
nlobjRequest.getParameter(name)	-	-	-
nlobjRequest.getParameterValues(name)	-	-	-
nlobjRequest.getURL()	-	-	-

nlobjResponse

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjResponse	-	-	-
nlobjResponse.addHeader(name, value)	-	-	-
nlobjResponse.getAllHeaders()	-	-	-
nlobjResponse.getBody()	-	-	-
nlobjResponse	-	-	-
nlobjResponse.getStatusCode()	-	-	-
nlobjResponse.getError()	-	-	-
nlobjResponse.getHeader(name)	-	-	-
nlobjResponse.getHeaders(name)	-	-	-
nlobjResponse.renderPDF(xmlString)	-	-	-
nlobjResponse.setCDNCacheable(type)	-	-	-
nlobjResponse.setContentType(type, name, disposition)	-	-	-
nlobjResponse.setEncoding(encodingType)	-	-	-
nlobjResponse.setHeader(name, value)	-	-	-
nlobjResponse.sendRedirect(type, identifier, id, editmode, parameters)	-	-	-
nlobjResponse.write(output)	-	-	-
nlobjResponse.writeLine(output)	-	-	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjResponse.writePage(pageobject)	-	-	-

nlobjSearch

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearch	search.Search	N/search Module	-
nlobjSearch.addColumn(column)	-	N/search Module	-
nlobjSearch.addColumns(columns)	-	N/search Module	-
nlobjSearch.addFilter(filter)	-	N/search Module	-
nlobjSearch.addFilters(filters)	-	N/search Module	-
nlobjSearch.deleteSearch()	search.delete(options)	N/search Module	For a script sample, see N/search Module Script Samples .
nlobjSearch.getColumns()	Search.columns	N/search Module	Note that <code>columns</code> is a property.
nlobjSearch.getFilterExpression()	Search.filterExpression	N/search Module	Note that <code>filterExpression</code> is a property.
nlobjSearch.getFilters()	Search.filters	N/search Module	Note that <code>filters</code> is a property.
nlobjSearch.getId()	Search.searchId	N/search Module	Note that <code>id</code> is a property.
nlobjSearch.getIsPublic()	Search.isPublic	N/search Module	Note that <code>isPublic</code> is a property.
nlobjSearch.getScriptId()	Search.id	N/search Module	-
nlobjSearch.getSearchType()	Search.searchType	N/search Module	Note that <code>searchType</code> is a property.
nlobjSearch.runSearch()	Search.run()	N/search Module	-
nlobjSearch.saveSearch(title, scriptId)	Search.save()	N/search Module	-
nlobjSearch.setColumns(columns)	Search.columns	N/search Module	Note that <code>columns</code> is a property.
nlobjSearch.setFilterExpression(filterExpression)	Search.filterExpression	N/search Module	Note that <code>filterExpression</code> is a property.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearch.setFilters(filters)	Search.filters	N/search Module	Note that <code>filters</code> is a property.
nlobjSearch.setIsPublic(type)	Search.isPublic	N/search Module	Note that <code>isPublic</code> is a property.
nlobjSearch.setRedirectURLToSearch()	redirect.toSavedSearch(options) redirect.toSearch(options)	N/redirect Module	-
nlobjSearch.setRedirectURLToSearchResult()	redirect.toSearchResult(options) redirect.toSavedSearchResult(options)	N/redirect Module	-

nlobjSearchColumn

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearchColumn	search.Column	N/search Module	-

nlobjSearchFilter

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearchFilter	search.Filter	N/search Module	-

nlobjSearchResult

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearchResult	search.Result	N/search Module	-

nlobjSearchResultSet

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSearchResultSet	search.ResultSet	N/search Module	-

nlobjSelectOption

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSelectOption	See Notes	N/record Module	See mapping for <code>nlobjSelectOption</code> methods.
nlobjSelectOption.getId()	N/record: Field.getSelectOptions(options) N/currentRecord: Field.getSelectOptions(options)	N/record Module N/currentRecord Module	-
nlobjSelectOption.getText()	N/record: Field.getSelectOptions(options)	N/record Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
	N/currentRecord: Field.getSelectOptions(options)	N/currentRecord Module	

nlobjSublist

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSubList	serverWidget.Sublist	N/ui/serverWidget Module	-
nlobjSublist.addButton(name, label, script)	Sublist.addButton(options)	N/ui/serverWidget Module	-
nlobjSublist.addField(name, type, label, source)	Sublist.addField(options)	N/ui/serverWidget Module	-
nlobjSublist.addMarkAllButtons()	Sublist.addMarkAllButtons()	N/ui/serverWidget Module	-
nlobjSublist.addRefreshButton()	Sublist.addRefreshButton()	N/ui/serverWidget Module	-
nlobjSublist.getLineItemCount()	Sublist.lineCount	N/ui/serverWidget Module	Note that lineCount is a property
nlobjSublist.getLineItemValue(group, fldnam, linenum)	Sublist.getSublistValue(options)	N/ui/serverWidget Module	-
nlobjSublist.setAmountField(field)	-	N/ui/serverWidget Module	-
nlobjSublist.setDisplayType(type)	Sublist.displayType	N/ui/serverWidget Module	Note that displayType is a property
nlobjSublist.setHelpText(help)	Sublist.helpText	N/ui/serverWidget Module	Note that helpText is a property
nlobjSublist.setLabel(label)	Sublist.label	N/ui/serverWidget Module	Note that label is a property
nlobjSublist.setLineItemValue(name, linenum, value)	Sublist.setSublistValue(options)	N/ui/serverWidget Module	-
nlobjSublist.setLineItemValues(values)		N/ui/serverWidget Module	-
nlobjSublist.setUniqueField(name)	Sublist.updateUniqueFieldId(options)	N/ui/serverWidget Module	Note that uniqueFieldId is a property

nlobjSubrecord

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjSubrecord	See Notes	N/record Module	SuiteScript 2.0 subrecords are returned as record.Record objects.

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
			Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlobjSubrecord.cancel()	See Notes	N/record Module	SuiteScript 2.0 subrecords are returned as record.Record objects. Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .
nlobjSubrecord.commit()	See Notes	N/record Module	This API does not have a SuiteScript 2.0 equivalent. SuiteScript 2.0 subrecords are returned as record.Record objects. Note that scripting subrecords in SuiteScript 2.0 is fundamentally different from scripting subrecords in SuiteScript 1.0. For additional information, see the SuiteScript 2.0 topics under SuiteScript 2.0 Scripting Subrecords .

nlobjTab

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjTab	serverWidget.Tab	N/ui/serverWidget Module	-
nlobjTab.setLabel(label)	Tab.label	N/ui/serverWidget Module	Note that <code>label</code> is a property
nlobjTab.setHelpText(help)	Tab.helpText	N/ui/serverWidget Module	Note that <code>helpText</code> is a property

nlobjTemplateRenderer

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjTemplateRenderer	render.TemplateRenderer	N/render Module	For a script sample, see N/render Module Script Sample .
nlobjTemplateRenderer.addRecord(var record)	render.TemplateRenderer.addRecord(options)	N/render Module	For a script sample, see N/render Module Script Sample .
nlobjTemplateRenderer.addSearchResults(var searchResult)	render.TemplateRenderer.addSearchResults(options)	N/render Module	For a script sample, see N/render Module Script Sample .
nlobjTemplateRenderer.renderToResponse()	render.TemplateRenderer.renderToResponse(options)	N/render Module	-
nlobjTemplateRenderer.renderToString()	render.TemplateRenderer.renderAsString()	N/render Module	-

SuiteScript 1.0 API	SuiteScript 2.0 API	SuiteScript 2.0 Module	Notes
nlobjTemplateRenderer.setTemplate(template)	TemplateRenderer.templateContent	N/render Module	For a script sample, see N/render Module Script Sample .

SuiteScript 2.0 Script Types

The following script types are supported:

Script Type	Description
SuiteScript 2.0 Bundle Installation Script Type	Bundle installation scripts fire triggers that execute as part of bundle installation, update, or uninstall. Trigger execution can occur either before install, after install, before update, after update, or before uninstall. These triggers automatically complete required setup, configuration, and data management tasks for the bundle. Bundle installation scripts execute on the server.
SuiteScript 2.0 Client Script Type	Client scripts run on individual forms, can be deployed globally, and are applied to entity and transaction record types. Global client scripts enable centralized management of scripts that can be applied to an entire record type. Client scripts execute in the user's browser.
SuiteScript 2.0 Map/Reduce Script Type	Map/reduce scripts provide a structured framework for processing a large number of records or a large amount of data. These scripts can be scheduled for submission or submitted on an on-demand basis. Map/reduce scripts execute on the server.
SuiteScript 2.0 Mass Update Script Type	Mass update scripts allows you to programmatically perform custom mass updates to update fields that are not available through general mass updates. These scripts can run complex calculations across many records. Mass update scripts execute on the server.
SuiteScript 2.0 Portlet Script Type	Portlet scripts create custom dashboard portlets. For example, you can use SuiteScript to create a portlet that is populated on-the-fly with company messages based on data within the system. Portlet scripts execute on the server.
SuiteScript 2.0 RESTlet Script Type	RESTlets can be used to define custom RESTful integrations to NetSuite. RESTlets execute on the server.
SuiteScript 2.0 Scheduled Script Type	Scheduled scripts can be scheduled for submission or submitted on an on-demand basis. Scheduled scripts execute on the server.
SuiteScript 2.0 Suitelet Script Type	Suitelets enable the creation of dynamic web content and build NetSuite-looking pages. Suitelets can be used to implement custom front and backends. Suitelets execute on the server.
SuiteScript 2.0 User Event Script Type	User Event scripts are triggered when users work with records and data changes in NetSuite as they create, open, update, or save records. These scripts customize the workflow and association between your NetSuite entry forms. These scripts can also be used for doing additional processing before records are entered or for validating entries based on other data in the system. User event scripts execute on the server.
SuiteScript 2.0 Workflow Action Script Type	Workflow action scripts allow you to create custom actions that are defined on a record in a workflow. Workflow action scripts execute on the server.

Best practices

- Always thoroughly test your code before using it on your live NetSuite data.
- Type all record, field, sublist, tab, and subtab IDs in lowercase in your SuiteScript code.

- Prefix all custom script IDs and deployment IDs with an underscore (_).
- Do not hard-code any passwords in scripts. The password and password2 fields are supported for scripting.
- If the same code is used across multiple forms, ensure that you test any changes in the code for **each** form that the code is associated with.
- Include proper error handling sequences in your script wherever data may be inconsistent, not available, or invalid for certain functions. For example, if your script requires a field value to validate another, ensure that the field value is available.
- Organize your code into reusable chunks. Many functions can be used in a variety of forms. Any reusable functions should be stored in a common library file and then called into specific event functions for the required forms as needed.
- Place all custom code and markup, including third party libraries, in your own namespace.



Important: Custom code must not be used to access the NetSuite DOM. Developers must use SuiteScript APIs to access NetSuite UI components.

- Use the built in Library functions whenever possible for reading/writing Date/Currency fields and for querying XML documents
- During script development, componentize your scripts, load them individually, and then test each one — inactivating all but the one you are testing when multiple components are tied to a single user event.
- When working with script type events, your function name should correspond with the event. For example, a pageInit event can be named PageInit or formAPageInit.
- Since name values can change, ensure that you use **static** ID values in your API calls where applicable.
- Although you can use any desired naming conventions for functions within your code, it is recommended that you use custom namespaces or unique prefixes for all your function names.
- Thoroughly comment your code. This practice helps with debugging and development and assists NetSuite support in locating problems if necessary.
- You must use the `runtime.getCurrentScript()` function in the runtime module to reference script parameters. For example, use the following code to obtain the value of a script parameter named `custscript_case_field`:

```
define(['N/runtime'], function(runtime) {
    function pageInit(context) {
        var strField = runtime.getCurrentScript().getParameter('SCRIPT', 'custscript_case_field');
        ...
    };
});
```

- Make sure that your script does not take a long time to execute. A script may execute for a long time if any or all of the following occur:
 - The script performs a large number of record operations without going over the usage limit.
 - The script causes a large number of user event scripts and/or workflows to execute.
 - The script performs database searches or updates that collectively take a long time to finish

Each server-side script type or application has a time limit for execution. This limit is not fixed and depends on the script type or application. If a single execution of a server-side script or application takes longer than the time limit for that script type or application, a `SSS_TIME_LIMIT_EXCEEDED` error is thrown. This error can also be thrown from a script that is executed by another script (for example, from a user event script that is executed by a scheduled script).

SuiteScript 2.0 Bundle Installation Script Type

Bundle installation scripts are specialized server scripts that perform processes in target accounts as part of a bundle installation, update, or uninstall. These processes include setup, configuration, and data management tasks that would otherwise have to be completed by account administrators.

Every bundle can include a bundle installation script that is automatically run when the bundle is installed, upgraded, or uninstalled. Each bundle installation script can contain triggers to be executed before install, after install, before update, after update, and after uninstall.

Bundle installation script failures terminate bundle installations, updates, or uninstalls. Bundle installation scripts can include their own error handling in addition to errors thrown by SuiteBundler and the SuiteScript engine. An error thrown by a bundle installation script returns an error code of **Installation Error** followed by the text defined by the script author.

For information about scripting with bundle installation scripts, see [SuiteScript 2.0 Bundle Installation Script Entry Points](#).

Bundle Installation Script Sample

The script sample performs the following tasks:

- Before the bundle installation and the bundle update, ensure that the Time Off Management feature is enabled in the target NetSuite account and warn the user if the feature is not enabled.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.0
 * @NScriptType BundleInstallationScript
 */
define(['N/runtime'], function(runtime) {
  function checkPrerequisites() {
    if (!runtime.isFeatureInEffect({
      feature: 'TIMEOFFMANAGEMENT'
    }))
      throw 'The TIMEOFFMANAGEMENT feature must be enabled. ' +
        'Please enable the feature and try again.';
  }
  return {
    beforeInstall: function beforeInstall(params) {
      checkPrerequisites();
    },
    beforeUpdate: function beforeUpdate(params) {
      checkPrerequisites();
    }
  };
});
```

SuiteScript 2.0 Bundle Installation Script Entry Points

Script Entry Point	
afterInstall	Executes after a bundle is installed for the first time in a target account.

Script Entry Point	
<code>afterUpdate</code>	Executes after a bundle in a target account is updated.
<code>beforeInstall</code>	Executes before a bundle is installed for the first time in a target account.
<code>beforeUninstall</code>	Executes before a bundle is uninstalled from a target account.
<code>beforeUpdate</code>	Executes before a bundle in a target account is updated.

afterInstall

Description	Executes after a bundle is installed for the first time in a target account.
Returns	void
Since	Version 2016 Release 1

Parameters

 **Note:** The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.version</code>	number	The version of the bundle that is being installed in the target account.	Version 2016 Release 1

afterUpdate

Description	Executes after a bundle in a target account is updated.
Returns	void
Since	Version 2016 Release 1

Parameters

 **Note:** The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.fromVersion</code>	number	The version of the bundle that is currently installed in the target account.	Version 2016 Release 1
<code>params.toVersion</code>	number	The version of the bundle that is being installed in the target account.	Version 2016 Release 1

beforeInstall

Description	Executes before a bundle is installed for the first time in a target account. Calls to scheduled scripts are not supported.
Returns	void
Since	Version 2016 Release 1

Parameters

Note: The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.version</code>	number	The version of the bundle that is being installed in the target account.	Version 2016 Release 1

beforeUninstall

Description	Executes before a bundle is uninstalled from a target account. Calls to scheduled scripts are not supported.		
Returns	void		
Since	Version 2016 Release 1		

Parameters

Note: The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.version</code>	number	The version of the bundle that is being uninstalled from the target account.	Version 2016 Release 1

beforeUpdate

Description	Executes before a bundle in a target account is updated. Calls to scheduled scripts are not supported.		
Returns	void		
Since	Version 2016 Release 1		

Parameters

Note: The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.fromVersion</code>	number	The version of the bundle that is currently installed in the target account.	Version 2016 Release 1
<code>params.toVersion</code>	number	The version of the bundle that is being installed in the target account.	Version 2016 Release 1

SuiteScript 2.0 Client Script Type

Client scripts are scripts that are executed by predefined event triggers in a browser. They can validate user-entered data and auto-populate fields or sublists at various form events. For details, see [SuiteScript 2.0 Client Script Entry Points and API](#).

Scripts can be run on most standard records, custom record types, and custom NetSuite pages such as Suitelets. See the help topic [SuiteScript Supported Records](#) for a list of NetSuite records that support SuiteScript.



Important: Client scripts only execute in edit mode. If you have a deployed client script with a `pageInit` entry point, that script does not execute when you view the form. It executes when you click **Edit**.

The following triggers can run a client script.

- Loading a form for editing
- Entering or changing a value in a field (before and after it is entered)
- Entering or changing a value in a field that sources another field
- Selecting a line item on a sublist
- Adding a line item (before and after it is entered)
- Saving a form

Record-level client scripts are executed after any existing form-based clients are run, and before any user event scripts are run.

See the help topic [Script Usage Unit Limits](#) for details about client script governance.



Tip: You can set the order in which client scripts execute on the Scripted Records page. See *Using the Scripted Records Page*.

For additional information about SuiteScript 2.0 client scripts, see the following:

- [SuiteScript 2.0 Client Script Reference](#)
 - [SuiteScript 2.0 Client Scripts on `currentRecord`](#)
 - [SuiteScript 2.0 Client Script Role Restrictions](#)
 - [SuiteScript 2.0 Remote Objects in Client Scripts](#)
 - [SuiteScript 2.0 Client Script Best Practices](#)
- [SuiteScript 2.0 Client Script Entry Points and API](#)
 - [fieldChanged](#)
 - [lineInit](#)
 - [pageInit](#)
 - [postSourcing](#)
 - [saveRecord](#)
 - [sublistChanged](#)
 - [validateDelete](#)
 - [validateField](#)
 - [validateInsert](#)
 - [validateLine](#)

SuiteScript Client Script Sample

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following sample shows a client script applied to a sales order form. The script performs the following tasks:

- When the form loads for editing, the `pageInit` event sets the customer field to a specific customer.
- When form changes are saved, the `saveRecord` event ensures that the customer field is set and at least one sales order item is listed.
- When editing the quantity of a sales order item, the `validateField` event ensures that the number is less than three.
- When selecting a sales order item, the `fieldChanged` event updates a memo field to indicate that the item was selected.
- When a specific sales order item is selected, the `postSourcing` event updates the price level for that particular item.
- When an existing partner is selected, the `lineInit` event changes the selected partner to a specific partner.
- When a partner is deleted, the `validateDelete` event updates a memo field to indicate that the partner was deleted.
- When adding a new partner or editing an existing partner, the `validateInsert` and `validateLine` events ensure that their contribution is set to 100%.

```

/**
 * @NApiVersion 2.x
 * @NScriptType ClientScript
 */
define(['N/error'],
function(error) {
    function pageInit(context) {
        if (context.mode !== 'create')
            return;
        var currentRecord = context.currentRecord;
        currentRecord.setValue({
            fieldId: 'entity',
            value: 107
        });
    }
    function saveRecord(context) {
        var currentRecord = context.currentRecord;
        if (!currentRecord.getValue({
            fieldId: 'entity'
        }) || currentRecord.getLineCount({
            sublistId: 'item'
        }) < 1)
            throw error.create({
                name: 'MISSING_REQ_ARG',
                message: 'Please enter all the necessary fields on the salesorder before saving'
            });
        return true;
    }
    function validateField(context) {
        var currentRecord = context.currentRecord;
        var sublistName = context.sublistId;
        var sublistFieldName = context.fieldId;
        var line = context.line;
        if (sublistName === 'item') {
            if (sublistFieldName === 'quantity') {
                if (currentRecord.getCurrentSublistValue({
                    sublistId: sublistName,

```

```

        fieldId: sublistFieldName
    }) < 3)
    currentRecord.setValue({
        fieldId: 'otherrefnum',
        value: 'Quantity is less than 3'
    });
    else
        currentRecord.setValue({
            fieldId: 'otherrefnum',
            value: 'Quantity accepted'
        });
    }
}
return true;
}

function fieldChanged(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    var sublistFieldName = context.fieldId;
    var line = context.line;
    if (sublistName === 'item' && sublistFieldName === 'item')
        currentRecord.setValue({
            fieldId: 'memo',
            value: 'Item: ' + currentRecord.getCurrentSublistValue({
                sublistId: 'item',
                fieldId: 'item'
            }) + ' is selected'
        });
}

function postSourcing(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    var sublistFieldName = context.fieldId;
    var line = context.line;
    if (sublistName === 'item' && sublistFieldName === 'item')
        if (currentRecord.getCurrentSublistValue({
            sublistId: sublistName,
            fieldId: sublistFieldName
        }) === '39')
            if (currentRecord.getCurrentSublistValue({
                sublistId: sublistName,
                fieldId: 'pricelevels'
            }) !== '1-1')
                currentRecord.setCurrentSublistValue({
                    sublistId: sublistName,
                    fieldId: 'pricelevels',
                    value: '1-1'
                });
}

function lineInit(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    if (sublistName === 'partners')
        currentRecord.setCurrentSublistValue({
            sublistId: sublistName,

```

```

        fieldId: 'partner',
        value: '55'
    });
}

function validateDelete(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    if (sublistName === 'partners')
        if (currentRecord.getCurrentSublistValue({
            sublistId: sublistName,
            fieldId: 'partner'
        }) === '55')
            currentRecord.setValue({
                fieldId: 'memo',
                value: 'Removing partner sublist'
            });
    return true;
}

function validateInsert(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    if (sublistName === 'partners')
        if (currentRecord.getCurrentSublistValue({
            sublistId: sublistName,
            fieldId: 'contribution'
        }) !== '100.0%')
            currentRecord.setCurrentSublistValue({
                sublistId: sublistName,
                fieldId: 'contribution',
                value: '100.0%'
            });
    return true;
}

function validateLine(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    if (sublistName === 'partners')
        if (currentRecord.getCurrentSublistValue({
            sublistId: sublistName,
            fieldId: 'contribution'
        }) !== '100.0%')
            currentRecord.setCurrentSublistValue({
                sublistId: sublistName,
                fieldId: 'contribution',
                value: '100.0%'
            });
    return true;
}

function sublistChanged(context) {
    var currentRecord = context.currentRecord;
    var sublistName = context.sublistId;
    var op = context.operation;
    if (sublistName === 'item')
        currentRecord.setValue({
            fieldId: 'memo',

```

```

        value: 'Total has changed to ' + currentRecord.getValue({
            fieldId: 'total'
        }) + ' with operation: ' + op
    });
}
return {
    pageInit: pageInit,
    fieldChanged: fieldChanged,
    postSourcing: postSourcing,
    sublistChanged: sublistChanged,
    lineInit: lineInit,
    validateField: validateField,
    validateLine: validateLine,
    validateInsert: validateInsert,
    validateDelete: validateDelete,
    saveRecord: saveRecord
};
});

```

SuiteScript 2.0 Client Script Reference

- [SuiteScript 2.0 Client Scripts on currentRecord](#)
- [SuiteScript 2.0 Client Script Role Restrictions](#)
- [SuiteScript 2.0 Remote Objects in Client Scripts](#)
- [SuiteScript 2.0 Client Script Best Practices](#)

SuiteScript 2.0 Client Scripts on currentRecord

You use the `currentRecord` module to access the record that is active in the current client-side context. For more information about the `currentRecord` module, see [N/currentRecord Module](#).

The following is a sample client script. When the `saveRecord` client event is triggered, the script uses the `currentRecord` module to validate a Sales Order record. The script ensures that the transaction date is not older than one week, and that the total is valid.

```

/**
 * @NApiVersion 2.x
 * @NScriptType ClientScript
 */
define(['N/ui/message'],
    function(msg) {
        function showErrorMessage(msgText) {
            var myMsg = msg.create({
                title: "Cannot Save Record",
                message: msgText,
                type: msg.Type.ERROR
            });

            myMsg.show({
                duration: 5000
            });
        }
    }
)

```

```

function saveRec(context) {
    var rec = context.currentRecord;
    var currentDate = new Date()
    var oneWeekAgo = new Date(currentDate - 1000 * 60 * 60 * 24 * 7);

    // Validate transaction date is not older than current time by one week
    if (rec.getValue({
        fieldId: 'trandate'
    }) < oneWeekAgo) {
        showErrorMessage("Cannot save sales order with trandate one week old.");
        return false;
    }

    //validate total is greater than 0
    if (rec.getValue({
        fieldId: 'total'
    }) <= 0) {
        showErrorMessage("Cannot save sales order with negative total amount.");
        return false;
    }
    return true;
}

return {
    saveRecord: saveRec
}
});

```

SuiteScript 2.0 Client Script Role Restrictions

Client scripts respect the role permissions specified in the user's NetSuite account. An error is thrown when running a client script to access a record with a role that does not have permission to view or edit the record.

The following client script attaches to a custom sales order form and executes on the field change client event:

```

/**
 * @NApiVersion 2.x
 * @NScriptType ClientScript
 */
define(['N/search'],
    function(search) {
        function(search) {
            function getSalesRepEmail(context) {
                var salesRep = context.currentRecord.getValue({
                    fieldId: 'salesrep'
                });
                var salesRepEmail = search.lookupFields({
                    type: 'employee',
                    id: salesRep,
                    columns: ['email']
                });
                alert(JSON.stringify(salesRepEmail));
            }
        }
    }
);

```

```

        return {
            fieldChanged: getSalesRepEmail
        }
    });

```

If you are logged in with an admin role, you receive the alert when you load the sales order with this form. If you are logged in with a role that does not have permission to view/edit Employee records, you receive an error when you select the Sales Rep field.

The following considerations can help prevent users from receiving the error:

- Consider the types of users who may be using your custom form and running the script.
- Consider which record types that users do not have access to. If it is vital that all who run the script have access to the records in the script, you may need to redefine the permissions of the users (if your role is as an admin).
- Consider rewriting your script so that it only references record types that all users have access to.
- Consider writing the script as a server-side user event script, and set the Execute As Admin preference on the Script Deployment page. Note that alerts are a function of client scripts only and cannot be used in user event scripts. For more information about user event scripts, see [SuiteScript 2.0 User Event Script Type](#).

SuiteScript 2.0 Remote Objects in Client Scripts

A client script interfaces with remote objects whenever it calls the NetSuite database to create, load, copy, or transform an object record.

The following client script loads a journal entry record and sets two line sublist lines when the saveRecord client event executes.

```

/**
 *@NApiVersion 2.0
 *@NScriptType ClientScript
 */
define(['N/record'], function(record){
    function saveRecord(context){
        var journalEntry = record.load({
            id:6,
            type: record.Type.JOURNAL_ENTRY,
            isDynamic: true
        });
        journalEntry.selectNewLine({
            sublistId: 'line'
        });
        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'account',
            value: 1
        });
        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'debit',
            value: 1
        });
    }
});

```

```

        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'memo',
            value: "Debit 1"
        });
        journalEntry.commitLine({
            sublistId: 'line'
        });
        journalEntry.selectNewLine({
            sublistId: 'line'
        });
        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'account',
            value: 1
        });
        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'credit',
            value: 1
        });
        journalEntry.setCurrentSublistValue({
            sublistId: 'line',
            fieldId: 'memo',
            value: "Credit 1"
        });
        journalEntry.commitLine({
            sublistId: 'line'
        });

        var recordId = journalEntry.save();
        return true;
    }
    return {
        saveRecord : saveRecord
    };
};

```

SuiteScript 2.0 Client Script Best Practices

The following list shows best practices for both form-level and record-level client script development:

- When testing form-level client scripts, ensure that the latest scripts are being executed. Use Ctrl-Refresh to clear your browser cache.
- Use global (record-level) client scripts to get a more flexible deployment model and to port (bundle) scripts.
- `record.setValue` and `record.setCurrentSublistValue` execution is multi-threaded whenever child field values need to be sourced in. To synchronize your logic, use the `postSourcing` function or set the `fireSlavingSync` synchronous parameter to `true`.
- Methods that set values accept raw data of a specific type and do not require formatting or parsing. Methods that set text accept strings but can conform to a user-specified format. For example, when setting a numerical field type, `setValue` accepts any number, and `setText` accepts a string in a specified format. (such as "2,000.39" or "2.00,39") When setting a Date field type, `setValue` accepts any Date object and `setText` accepts a string in a specified format. (such as "6/9/2016" or "9/6/2016")

SuiteScript 2.0 Client Script Entry Points and API

Script Entry Point	Description
<code>fieldChanged</code>	Executed when a field is changed by a user or client side call.
<code>lineInit</code>	Executed when an existing line is selected.
<code>pageInit</code>	Executed when the page completes loading or when the form is reset.
<code>postSourcing</code>	Executed on transaction forms when a field that sources information from another field is modified.
<code>saveRecord</code>	Executed after the submit button is pressed but before the form is submitted.
<code>sublistChanged</code>	Executed after a sublist has been inserted, removed, or edited.
<code>validateDelete</code>	Executed when removing an existing line from an edit sublist.
<code>validateField</code>	Executes when a field is about to be changed by a user or client side call.
<code>validateInsert</code>	Executed when you insert a line into an edit sublist.
<code>validateLine</code>	Executed before a line is added to an inline editor sublist or editor sublist.

fieldChanged

Description	<p>Executed when a field is changed by a user or client side call. This event may also execute directly through <code>beforeLoad</code> user event scripts. The following sample tasks can be performed:</p> <ul style="list-style-type: none"> Provide the user with additional information based on user input. Disable or enable fields based on user input. <p>For an example, see SuiteScript Client Script Sample.</p> <div>  Note: This event does not execute when the field value is changed or entered in the page URL. Use the <code>pageInit</code> function to handle URLs that may contain updated field values. See pageInit. </div>
Returns	void
Since	Version 2015 Release 2

Parameters

 Note: The <code>scriptContext</code> parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	<code>currentRecord.CurrentRecord</code>	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015

Parameter	Type	Description	Since
			Release 2
<code>scriptContext.fieldId</code>	string	The field ID name.	Version 2015 Release 2
<code>scriptContext.line</code>	string	The line number (zero-based index) if the field is in a sublist or a matrix. If the field is not a sublist or matrix, the default value is undefined.	Version 2015 Release 2
<code>scriptContext.column</code>	string	The column number (zero-based index) if the field is in a matrix. If the field is not in a matrix, the default value is undefined.	Version 2015 Release 2

lineInit

Description	Executed when an existing line is selected. This event can behave like a <code>pageInit</code> event for line items in an inline editor sublist or editor sublist. For an example, see SuiteScript Client Script Sample .
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2

pageInit

Description	Executed when the page completes loading or when the form is reset. The following sample tasks can be performed:
--------------------	--

	<ul style="list-style-type: none"> ■ Populate field defaults. ■ Disable or enable fields. ■ Change field availability or values depending on the data available for the record. ■ Add flags to set initial values of fields. ■ Provide alerts where the data being loaded is inconsistent or corrupt. ■ Retrieve user login information and change field availability or values accordingly. ■ Validate that fields required for your custom code (but not necessarily required for the form) exist. <p>For an example, see SuiteScript Client Script Sample.</p>
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.mode</code>	string	The mode in which the record is being accessed. The mode can be set to one of the following values: <ul style="list-style-type: none"> ■ copy ■ create ■ edit 	Version 2015 Release 2

Note: For a complete script example, see [SuiteScript Client Script Sample](#).

postSourcing

Description	<p>Executed on transaction forms when a field that sources information from another field is modified.</p> <p>This event behaves like a <code>fieldChanged</code> event after all dependent field values have been set. The event waits for any slaved or cascaded field changes to complete before calling the user defined function.</p> <p>For an example, see SuiteScript Client Script Sample.</p> <div> <p>Note: The event is not triggered by field changes for a field that does not have any slaved fields.</p> </div>
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2
<code>scriptContext.fieldId</code>	string	The field ID name.	Version 2015 Release 2

saveRecord

Description	Executed after the submit button is pressed but before the form is submitted. The following sample tasks can be performed: <ul style="list-style-type: none"> Provide alerts before committing the data. Enable fields that were disabled with other functions. Redirect the user to a specified URL. For an example, see SuiteScript Client Script Sample .
Returns	Boolean <code>true</code> if the record is valid. Boolean <code>false</code> to suppress form submission.
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2

sublistChanged

Note: The `sublistChanged` function replaces the SuiteScript 1.0 function `recalc`.

Description	Executed after a sublist is inserted, removed, or edited.
--------------------	---

	For an example, see SuiteScript Client Script Sample .
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2

validateDelete

Description	Executed when removing an existing line from an edit sublist. For an example, see SuiteScript Client Script Sample .
Returns	Boolean <code>true</code> if the sublist line is valid. Boolean <code>false</code> to block the removal.
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	currentRecord.CurrentRecord	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2

validateField

Description	Executes when a field is about to be changed by a user or client side call. This event executes on fields added in <code>beforeLoad</code> user event scripts. The following sample tasks can be performed:
--------------------	---

	<ul style="list-style-type: none"> Validate field lengths. Restrict field entries to a predefined format. Restrict submitted values to a specified range Validate the submission against entries made in an associated field. <p>For an example, see SuiteScript Client Script Sample.</p> <div>  Note: This event does not apply to dropdown select or check box fields. </div>
Returns	Boolean <code>true</code> if the field is valid. Boolean <code>false</code> to prevent the field value from changing.
Since	Version 2015 Release 2

Parameters

 **Note:** The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	<code>currentRecord.CurrentRecord</code>	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2
<code>scriptContext.fieldId</code>	string	The field ID name.	Version 2015 Release 2
<code>scriptContext.line</code>	string	The line number (zero-based index) if the field is in a sublist or a matrix. If the field is not a sublist or matrix, the default value is undefined.	Version 2015 Release 2
<code>scriptContext.column</code>	string	The column number (zero-based index) if the field is in a matrix. If the field is not in a matrix, the default value is undefined.	Version 2015 Release 2

validateInsert

Description	Executed when you insert a line into an edit sublist. For an example, see SuiteScript Client Script Sample .
Returns	Boolean <code>true</code> if the sublist line is valid. Boolean <code>false</code> to block the insert.
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	<code>currentRecord.CurrentRecord</code>	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2

validateLine

Description	Executed before a line is added to an inline editor sublist or editor sublist. This event can behave like a <code>saveRecord</code> event for line items in an inline editor sublist or editor sublist. For an example, see SuiteScript Client Script Sample .
Returns	Boolean <code>true</code> if the sublist line is valid. Boolean <code>false</code> to reject the operation.
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>scriptContext.currentRecord</code>	<code>currentRecord.CurrentRecord</code>	The current form record. For more information about <code>CurrentRecord</code> object members, see CurrentRecord Object Members .	Version 2015 Release 2
<code>scriptContext.sublistId</code>	string	The sublist ID name.	Version 2015 Release 2

SuiteScript 2.0 Map/Reduce Script Type

The map/reduce script type is designed for scripts that need to handle large amounts of data. It is best suited for situations where the data can be divided into small, independent parts. When the script is executed, a structured framework automatically creates enough jobs to process all of these parts. You as the user do not have to manage this process. Another advantage of map/reduce is that these jobs can work in parallel. You choose the level of parallelism when you deploy the script.

Like a scheduled script, a map/reduce script can be invoked manually or on a predefined schedule. However, map/reduce scripts offer several advantages over scheduled scripts. One advantage is that, if a map/reduce job violates certain aspects of NetSuite governance, the map/reduce framework automatically causes the job to yield and its work to be rescheduled for later, without disruption to the script. However, be aware that some aspects of map/reduce governance cannot be handled through automatic yielding. For that reason, if you use this script type, you should familiarize yourself with the [SuiteScript 2.0 Map/Reduce Governance](#) guidelines.

In general, you should use map/reduce for any scenario where you want to process multiple records, and where your logic can be separated into relatively lightweight segments. In contrast, map/reduce is not as well suited to situations where you want to enact a long, complex function for each part of your data set. A complex series of steps might be one that includes the loading and saving of multiple records.

For more information about map/reduce scripts, see the following topics:

- [Map/Reduce Use Cases](#)
- [Map/Reduce Key Concepts](#)
- [Map/Reduce Entry Points](#)
- [Map/Reduce Script Samples](#)

Map/Reduce Use Cases

Map/reduce is ideal for scenarios where you want to apply the same logic repeatedly — once for each object in a series. For example, you could use a map/reduce script to do any of the following:

- Identify a list of requisitions and transform each requisition into a purchase order.
- Search for invoices that meet certain criteria and apply a discount to each one.
- Search for customer records that appear to be duplicates, then process each apparent duplicate according to your business rules.
- Search for outstanding tasks assigned to sales reps, then send each person an email that summarizes their outstanding work.
- Identify files in the NetSuite File Cabinet, use the content of the files to create new documents, and upload the new documents to an external server.

Map/Reduce Key Concepts

Inspired by the [map/reduce paradigm](#), the general idea behind a map/reduce script is as follows: Your script identifies some data that requires processing. This data is split into key/value pairs. Your script defines a function that the system invokes one time for each key/value pair. Optionally, your script can also use a second round of processing. Depending on how you deploy the script, the system can create multiple jobs for each round of processing and process the data in parallel.

If you are familiar with other SuiteScript 2.0 script types, then you may already recognize that map/reduce scripts are significantly different from most types. Before you begin writing a map/reduce script, make sure you understand these differences. Consider the following:

- [Map/reduce scripts are executed in stages](#)
- [The system supplements your logic](#)
- [The system provides robust context objects](#)
- [Multiple jobs are used to execute one script](#)
- [Map/reduce scripts permit yielding and other interruptions](#)

Map/reduce scripts are executed in stages

With most script types, each script is executed as a single continuous process. In contrast, a map/reduce script is executed in five discrete stages. These stages occur in a specific sequence.

You can control the script's behavior in four of the five stages. That is, each of these four stages corresponds to an entry point. Your corresponding function defines the script's behavior during that stage. For example:

- For the **getInputData** stage, you write a function that returns an object that can be transformed into a list of key/value pairs. For example, if your function returns a search of NetSuite records, the system would run the search. The key/value pairs would be the results of the search: Each key would be the internal ID of a record. Each value would be a JSON representation of the record's field IDs and values.
- For the **map** stage, you can optionally write a function that the system invokes one time for each key/value pair. If appropriate, your map function can write output data, in the form of new key/value pairs. If the script also uses a reduce function, this output data is sent as input to the shuffle and then the reduce stage. Otherwise, the new key/value pairs are sent directly to the summarize stage.
- You do not write a function for the **shuffle** stage. In this stage, the system sorts through any key/value pairs that were sent to the reduce stage, if a reduce function is defined. These pairs may have been provided by the map function, if a map function is used. If a map function was not used, the shuffle stage uses data provided by the getInputData stage. The shuffle stage groups this data by key to form a new set of key/value pairs, where each key is unique and each value is an array. For example, suppose 100 key/value pairs were sent. Suppose that each key represents an employee, and each value represents a record that the employee created. If there were only two unique employees, and one employee created 90 records, while another employee created 10 records, then the shuffle stage would provide two key/value pairs. The keys would be the IDs of the employees. One value would be an array of 90 elements, and the other would be an array of 10 elements.
- For the **reduce** stage, you write a function that is invoked one time for each key/value pair that was provided by the shuffle stage. Optionally, this function can write data as key/value pairs that are sent to the summarize stage.
- In the **summarize** stage, your function can retrieve and log statistics about the script's work. It can also take actions with data sent by the reduce stage.

Note that you may omit either the map or reduce function. You can also omit the summarize function. For more details, review [Map/Reduce Entry Points](#).

The system supplements your logic

With most script types, the functionality of the script is determined entirely by the code within your script file. Map/reduce is different. With map/reduce, the logic in your file is important, but the system also supplements your logic with standardized logic of its own. For example, the system moves data between the stages. Additionally, the system invokes your map and reduce function multiple times. For this reason, think of the logic of the map and reduce functions as being similar to the logic you would use in a loop. Each of these functions should perform a relatively small amount of work. For details about the system's behavior during and between the stages, see [SuiteScript 2.0 Map/Reduce Script Stages](#).

The system provides robust context objects

For each entry point function you write, the system provides a context object. In itself, this fact is not groundbreaking – the system makes a context object available to most SuiteScript 2.0 entry points. However, the objects provided to map/reduce entry point functions are especially robust. These objects contain data and properties that are critical to writing an effective script. For example, you can use these objects to access data from the previous stage and write output data that is sent to the next

stage. Context objects can also contain data about errors, usage units consumed, and other statistics. For details, see [SuiteScript 2.0 Map/Reduce Script Entry Points and API](#).

Multiple jobs are used to execute one script

All map/reduce scripts are powered by [SuiteCloud Processors](#), which handle work through a series of **jobs**. Each job is executed by a **processor**, which is a virtual unit of processing power. SuiteCloud Processors are also used to process scheduled scripts. However, these two script types are handled differently. For example, the system always creates only one job to handle a scheduled script. In contrast, the system creates multiple jobs to process a single map/reduce script. Specifically, the system creates at least one job to execute each stage. Additionally, multiple jobs can be created to handle the work of the map stage, and multiple jobs can be created for the reduce stage. When the system creates multiple map and reduce jobs, these jobs work independently of each other and may work in parallel across multiple processors. For this reason, the map and reduce stages are considered **parallel** stages.

In contrast, the `getInputData` and `summarize` stages are each executed by one job. In each case, that job invokes your function only one time. These stages are **serial** stages. The shuffle stage is also a serial stage.

Map/reduce scripts permit yielding and other interruptions

Since the map and reduce stages consist of multiple independent map and reduce function invocations, the work of these stages can easily be divided among multiple jobs. The structure is naturally flexible. It allows for parallel processing, and it also permits map and reduce jobs to manage some aspects of their own resource consumption.

If a job monopolizes a processor for too long, the system can naturally finish the job after the current map or reduce function has completed. In this case, the system creates a new job to continue executing remaining key/value pairs. Based on its priority and submission timestamp, the new job either starts right after the original job has finished, or it starts later, to allow higher-priority jobs processing other scripts to execute. For more details, see [SuiteScript 2.0 Map/Reduce Yielding](#).

At the same time, be aware that the system imposes some usage limits on map/reduce scripts that are not managed through yielding. For details, see [SuiteScript 2.0 Map/Reduce Governance](#).

Map/Reduce Entry Points

A map/reduce script can go through a total of five stages. One stage, shuffle, does not correspond with an entry point. The other stages do. Their entry points are described in the following table.

Entry point	Purpose of corresponding function	Required?
<code>getInputData(inputContext)</code>	To identify data that needs processing. The system passes this data to the next stage.	yes
<code>map(mapContext)</code>	To apply processing to each key/value pair provided, and optionally pass data to the next stage.	One of these two entry points is required. You can also use both entry points.
<code>reduce(reduceContext)</code>	To apply processing to each key/value pair provided. In this stage, each key is unique, and each value is an array of values. This function can optionally pass data to the summarize stage.	
<code>summarize(summaryContext)</code>	To retrieve data about the script's execution, and take any needed actions with the output of the reduce stage.	no

For full details on the map/reduce entry points and their corresponding context objects, see [SuiteScript 2.0 Map/Reduce Script Entry Points and API](#).

Map/Reduce Script Samples

See the following samples:

- [Example: Script that Counts Characters](#)
- [Example: Script that Processes Invoices](#)

Example: Script that Counts Characters

The following sample is a very simple map/reduce script. This sample does not accomplish a realistic business objective but rather is designed to demonstrate how the script type works.

This script defines a hardcoded string. The script counts the number of times that each letter of the alphabet occurs within the string. Then it creates a file that shows its results. Refer to the comments in the sample for details about how the system processes the script.

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

define(['N/file'],

function(file) {

    // Define characters that should not be counted when the script performs its
    // analysis of the text.

    const PUNCTUATION_REGEXP = /[\\u2000-\\u206F\\u2E00-\\u2E7F\\'\\!\"#$%&\\
    (\\)\\*\\+\\,\\-\\.\\/:\\;<=>\\?@[\\]\\^\\'\\{\\|\\}\\~]/g;

    // Use the getInputData function to return two strings.

    function getInputData() {
        return "the quick brown fox \\njumps over the lazy dog.".split('\\n');
    }

    // After the getInputData function is executed, the system creates the following
    // key/value pairs:
    //
    // key: 0, value: 'the quick brown fox'
    // key: 1, value: 'jumps over the lazy dog.'

    // The map function is invoked one time for each key/value pair. Each time the
    // function is invoked, the relevant key/value pair is made available through
    // the context.key and context.value properties.

    function map(context) {

        // Create a loop that examines each character in the string. Exclude spaces
        // and punctuation marks.
```

```

for (var i = 0; context.value && i < context.value.length; i++)
    if (context.value[i] != ' ' && !PUNCTUATION_REGEXP.test(context.value[i])) {

        // For each character, invoke the context.write() method. This method saves
        // a new key/value pair. For the new key, save the character currently being
        // examined by the loop. For each value, save the number 1.

        context.write({
            key: context.value[i],
            value: 1
        });
    }
}

// After the map function has been invoked for the last time, the shuffle stage
// begins. In this stage, the system sorts the 35 key/value pairs that were saved
// by the map function during its two invocations. From those pairs, the shuffle
// stage creates a new set of key/value pairs, where the each key is unique. In
// this way, the number of key/value pairs is reduced to 25. For example, the map
// stage saved three instances of {'e','1'}. In place of those pairs, the shuffle
// stage creates one pair: {'e', ['1','1','1']}. These pairs are made available to
// the reduce stage through the context.key and context.values properties.

// The reduce function is invoked one time for each of the 25 key/value pairs
// provided.

function reduce(context) {

    // Use the context.write() method to save a new key/value pair, where the new key
    // equals the key currently being processed by the function. This key is a letter
    // in the alphabet. Make the value equal to the length of the context.values array.
    // This number represents the number of times the letter occurred in the original
    // string.

    context.write({
        key: context.key,
        value: context.values.length
    });
}

// The summarize stage is a serial stage, so this function is invoked only one
// time.

function summarize(context) {

    // Log details about the script's execution.

    log.audit({
        title: 'Usage units consumed',
        details: context.usage
    });

    log.audit({
        title: 'Concurrency',

```

```

        details: context.concurrency
    });

    log.audit({
        title: 'Number of yields',
        details: context.yields
    });

    // Use the context object's output iterator to gather the key/value pairs saved
    // at the end of the reduce stage. Also, tabulate the number of key/value pairs
    // that were saved. This number represents the total number of unique letters
    // used in the original string.

    var text = '';
    var totalKeysSaved = 0;
    context.output.iterator().each(function(key, value) {
        text += (key + ' ' + value + '\n');
        totalKeysSaved++;
        return true;
    });

    // Log details about the total number of pairs saved.

    log.audit({
        title: 'Unique number of letters used in string',
        details: totalKeysSaved
    });

    // Use the N/file module to create a file that stores the reduce stage output,
    // which you gathered by using the output iterator.

    var fileObj = file.create({
        name: 'letter_count_result.txt',
        fileType: file.Type.PLAINTEXT,
        contents: text
    });

    fileObj.folder = -15;
    var fileId = fileObj.save();

    log.audit({
        title: 'Id of new file record',
        details: fileId
    });
}

// Link each entry point to the appropriate function.

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

```

Example: Script that Processes Invoices

The following example shows a sample script that processes invoices and contains logic to handle errors. This script is designed to do the following:

- Find the customers associated with all open invoices.
- Apply a location-based discount to each invoice.
- Write each invoice to the reduce stage so it is grouped by customer.
- Initialize a new CustomerPayment for each customer applied only to the invoices specified in the reduce values.
- Create a custom record capturing the details of the records that were processed.
- Notify administrators of any exceptions via an email notification.

Prior to running this sample, you need to manually create a custom record type with id "customrecord_summary", and text fields with id "custrecord_time", "custrecord_usage", and "custrecord_yields".

Script Sample Prerequisites

1. From the NetSuite UI, select Customization > List, Records, & Fields > Record Types > New.
2. From the Custom Record Type page, enter a value for name.
3. In the ID field, enter "customrecord_summary".
4. Select Save.
5. From the Fields subtab, do the following:
 - Select New Field. Enter a label and set ID to "custrecord_time". Ensure that the Type field is set to Free-Form Text. Select Save & New.
 - Select New Field. Enter a label and set ID to "custrecord_usage". Ensure that the Type field is set to Free-Form Text. Select Save & New.
 - Select New Field. Enter a label and set ID to "custrecord_yields". Ensure that the Type field is set to Free-Form Text. Select Save.

```
/**
 * @NApiVersion 2.0
 * @NScriptType MapReduceScript
 */
define(['N/search', 'N/record', 'N/email', 'N/runtime', 'N/error'],
function(search, record, email, runtime, error)
{
    function handleErrorAndSendNotification(e, stage)
    {
        log.error('Stage: ' + stage + ' failed', e);

        var author = -5;
        var recipients = 'notify@company.com';
        var subject = 'Map/Reduce script ' + runtime.getCurrentScript().id + ' failed for stage: ' + stage;
        var body = 'An error occurred with the following information:\n' +
            'Error code: ' + e.name + '\n' +
            'Error msg: ' + e.message;

        email.send({
            author: author,
            recipients: recipients,
            subject: subject,
```

```

        body: body
    });
}

function handleErrorIfAny(summary)
{
    var inputSummary = summary.inputSummary;
    var mapSummary = summary.mapSummary;
    var reduceSummary = summary.reduceSummary;

    if (inputSummary.error)
    {
        var e = error.create({
            name: 'INPUT_STAGE_FAILED',
            message: inputSummary.error
        });
        handleErrorAndSendNotification(e, 'getInputData');
    }

    handleErrorInStage('map', mapSummary);
    handleErrorInStage('reduce', reduceSummary);
}

function handleErrorInStage(stage, summary)
{
    var errorMsg = [];
    summary.errors.iterator().each(function(key, value){
        var msg = 'Failure to accept payment from customer id: ' + key + '. Error was: ' +
JSON.parse(value).message + '\n';
        errorMsg.push(msg);
        return true;
    });
    if (errorMsg.length > 0)
    {
        var e = error.create({
            name: 'RECORD_TRANSFORM_FAILED',
            message: JSON.stringify(errorMsg)
        });
        handleErrorAndSendNotification(e, stage);
    }
}

function createSummaryRecord(summary)
{
    try
    {
        var seconds = summary.seconds;
        var usage = summary.usage;
        var yields = summary.yields;

        var rec = record.create({
            type: 'customrecord_summary',
        });

        rec.setValue({

```

```

        fieldId : 'name',
        value: 'Summary for M/R script: ' + runtime.getCurrentScript().id
    });

    rec.setValue({
        fieldId: 'custrecord_time',
        value: seconds
    });
    rec.setValue({
        fieldId: 'custrecord_usage',
        value: usage
    });
    rec.setValue({
        fieldId: 'custrecord_yields',
        value: yields
    });

    rec.save();
}
catch(e)
{
    handleErrorAndSendNotification(e, 'summarize');
}
}

function applyLocationDiscountToInvoice(recordId)
{
    var invoice = record.load({
        type: record.Type.INVOICE,
        id: recordId,
        isDynamic: true
    });

    var location = invoice.getText({
        fieldId: 'location'
    });

    var discount;
    if (location === 'East Coast')
        discount = 'Eight Percent';
    else if (location === 'West Coast')
        discount = 'Five Percent';
    else if (location === 'United Kingdom')
        discount = 'Nine Percent';
    else
        discount = '';

    invoice.setText({
        fieldId: 'discountitem',
        text: discount,
        ignoreFieldChange : false
    });
    log.debug(recordId + ' has been updated with location-based discount.');
```

```

    invoice.save();
}
}
```



```

function getInputData()
{
    return search.create({
        type: record.Type.INVOICE,
        filters: [['status', search.Operator.IS, 'open']],
        columns: ['entity'],
        title: 'Open Invoice Search'
    });
}

function map(context)
{
    var searchResult = JSON.parse(context.value);
    var invoiceId = searchResult.id;
    var entityId = searchResult.values.entity.value;

    applyLocationDiscountToInvoice(invoiceId);

    context.write({
        key: entityId,
        value: invoiceId
    });
}

function reduce(context)
{
    var customerId = context.key;

    var custPayment = record.transform({
        fromType: record.Type.CUSTOMER,
        fromId: customerId,
        toType: record.Type.CUSTOMER_PAYMENT,
        isDynamic: true
    });

    var lineCount = custPayment.getLineCount('apply');
    for (var j = 0; j < lineCount; j++)
    {
        custPayment.selectLine({
            sublistId: 'apply',
            line: j
        });
        custPayment.setCurrentSublistValue({
            sublistId: 'apply',
            fieldId: 'apply',
            value: true
        });
    }

    var custPaymentId = custPayment.save();

    context.write({
        key: custPaymentId
    });
}

```

```

    });
  }

  function summarize(summary)
  {
    handleErrorIfAny(summary);
    createSummaryRecord(summary);
  }

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

```

SuiteScript 2.0 Map/Reduce Script Reference

- [SuiteScript 2.0 Map/Reduce Script Stages](#)
- [SuiteScript 2.0 Map/Reduce Terminology](#)
- [SuiteScript 2.0 Map/Reduce Yielding](#)
- [SuiteScript 2.0 Map/Reduce Script Submission](#)
- [SuiteScript 2.0 Map/Reduce Governance](#)
- [SuiteScript 2.0 Map/Reduce Script Status Page](#)
- [SuiteScript 2.0 Map/Reduce Script Error Handling](#)
- [SuiteScript 2.0 Map/Reduce Script Best Practices](#)
- [Adding Logic to Handle Map/Reduce Restarts](#)

SuiteScript 2.0 Map/Reduce Terminology

Term	Definition	More Information
buffer size	An option on the script deployment record. This choice determines the number of key/value pairs that a map or reduce job can process before information about the job's progress is saved to the database. As a general best practice, leave this value set to 1.	<ul style="list-style-type: none"> ■ Buffer Size
deployment instance	A task created to process a script deployment. A deployment instance is created when the deployment is submitted for processing. Only one unfinished instance of a particular script deployment can exist at any time.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Submission
exitOnError	A configuration option that affects the behavior of a script when an uncaught error interrupts a map or reduce function. When exitOnError is set to true, and all available retries have been used, the script exits the current stage and goes to the summarize stage.	<ul style="list-style-type: none"> ■ exitOnError

Term	Definition	More Information
function invocation	An execution of a function. Some map/reduce entry point functions, such as the <code>getInputData</code> and <code>summarize</code> functions, are invoked only one time during the script's processing. Others, such as the <code>map</code> and <code>reduce</code> functions, are invoked multiple times. For example, if the <code>getInputData</code> stage provides 20 key/value pairs to the map stage, then the map function is invoked 20 times, one time for each pair.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Type
<code>getInputData</code>	The first stage in the processing of a map/reduce script. During this stage, your script must return an object that can be transformed into a list of key/value pairs. You use the <code>getInputData</code> entry point to identify the function that executes during this stage.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages
governance	A system of rules governing your usage of NetSuite. Specific limits exist for every script type, including map/reduce.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Governance ■ SuiteScript Governance
hard limit	A type of map/reduce governance limit that, when violated, causes an interruption to the current function invocation. Make sure you follow the SuiteScript 2.0 Map/Reduce Script Best Practices to avoid problems with hard limits.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Governance ■ SuiteScript 2.0 Map/Reduce Script Best Practices
job	A unit of execution managed by SuiteCloud Processors . The processing of a map/reduce script is always handled by multiple jobs. At least one job is created to handle the processing of each stage.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages ■ SuiteCloud Processors
map	The second stage in the processing of a map/reduce script. A map function is optional, but your script must use either a map or reduce function (or both). When used, a map function processes data provided by the <code>getInputData</code> stage. The map function is invoked one time for each key/value pair that is provided.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages ■ map(mapContext)
map/reduce	A computing paradigm designed for the processing of large data sets.	<ul style="list-style-type: none"> ■ https://en.wikipedia.org/wiki/MapReduce
map/reduce script type	A SuiteScript 2.0 script type based on the map/reduce paradigm.	<ul style="list-style-type: none"> ■ Map/Reduce Key Concepts
parallel stage	A type of stage that can be handled by multiple jobs that work simultaneously. The map and reduce stages are parallel stages. See also serial stage.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages
priority	A property of a job. Job priority determines the order in which the scheduler sends jobs to the processor pool. Priorities can be set on the deployment record or on the SuiteCloud Processors – Priority Settings Page .	<ul style="list-style-type: none"> ■ Map/Reduce Script Deployment Record ■ SuiteCloud Processors – Priority Levels

Term	Definition	More Information
processor	In the context of SuiteCloud Processors, a virtual unit of processing power that executes a job. A processor is not an individual physical entity but rather a single processing thread.	<ul style="list-style-type: none"> ■ Concurrency Limit ■ SuiteCloud Processors
processor pool	In the context of SuiteCloud Processors, the total processors available to your NetSuite account. The number of processors available varies depending on your licensing. The number of processors available to process a particular script is determined by the value you choose for the Concurrency Limit field on the script deployment record.	<ul style="list-style-type: none"> ■ Concurrency Limit ■ SuiteCloud Processors
reduce	The fourth stage in the processing of a map/reduce script. A reduce function is optional, but your script must use either a map or reduce function (or both). When used, a reduce function processes data provided by the getInputData stage or, if a map function is used, by the map stage. The reduce function is invoked one time for each unique key from the list of all key/value pairs provided. The value for each of the unique keys passed to the reduce function is an array of all values with that key.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages ■ reduce(reduceContext)
retryCount	A configuration option that is relevant when a map or reduce function is interrupted by an uncaught error or by an application server restart. This setting determines whether the system invokes the map or reduce function again for any key/value pairs that were left in an uncertain state following the interruption.	<ul style="list-style-type: none"> ■ retryCount
script deployment	A set of configuration choices for how to execute a script. These choices are stored on a script deployment record. To execute a map/reduce script, you must create a deployment record and submit the deployment.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Submission
serial stage	A type of stage that is processed in its entirety by only one job. The getInputData, shuffle, and summarize stages are serial stages.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages
shuffle	The third stage in the processing of a map/reduce script. This stage is significant if your script uses a reduce function. In these cases, the shuffle stage sorts data provided by the getInputData stage or, if a map function is used, by the map stage. In either case, the shuffle stage creates a set of key/value pairs where each key is unique, and each value is an array. These pairs are passed to the reduce stage for further processing.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages

Term	Definition	More Information
soft limit	A type of map/reduce governance rule that, after it is surpassed, causes a map or reduce job to automatically yield. When the job yields, its work is rescheduled. Progress toward a soft limit is checked only after a function invocation has completed. Therefore this limit never causes an interruption to the function invocation.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Governance
SuiteCloud Processors	The engine that processes map/reduce and scheduled scripts.	<ul style="list-style-type: none"> ■ SuiteCloud Processors
summarize	The final stage in the processing of a map/reduce script. You can use this stage to log data about the work of the script.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Stages ■ <code>summarize(summaryContext)</code>
task	The set of work that represents a map/reduce deployment that has been submitted for processing. Each task is executed by a minimum of five jobs, one for each stage.	<ul style="list-style-type: none"> ■ SuiteScript 2.0 Map/Reduce Script Submission
yielding	A behavior that allows a map/reduce script to manage some aspects of its own resource consumption. After a map/reduce job has surpassed a soft limit, it automatically yields, and its work is rescheduled for later. Yielding never interrupts a function invocation. Note also that this process is automatic. SuiteScript 2.0 does not contain an API that lets you force a script to yield.	<ul style="list-style-type: none"> ■ Yield After Minutes

SuiteScript 2.0 Map/Reduce Script Stages

The map/reduce script type goes through at least two of five possible stages.

The stages are processed in the following order. Note that each stage must complete before the next stage begins.

- **Get Input Data** – Acquires a collection of data. This stage is always processed first and is required. The input stage runs sequentially.
- **Map** – Parses each row of data into a key/value pair. One pair (key/value) is passed per function invocation. If this stage is skipped, the reduce stage is required. Data may be processed in parallel in this stage.
- **Shuffle** – Groups values based on keys. This is an automatic process that always follows completion of the map stage. There is no direct access to this stage as it is handled by the map/reduce script framework. Data is processed sequentially in this stage.
- **Reduce** – Evaluates the data in each group. One group (key/values) is passed per function invocation. If this stage is skipped, the map stage is required. Data is processed in parallel in this stage.
- **Summarize** – Summarizes the output of the previous stages. Developers can use this stage to summarize the data from the entire map/reduce process and write it to a file or send an email. This stage is optional and is not technically a part of the map/reduce process. The summarize stage runs sequentially.

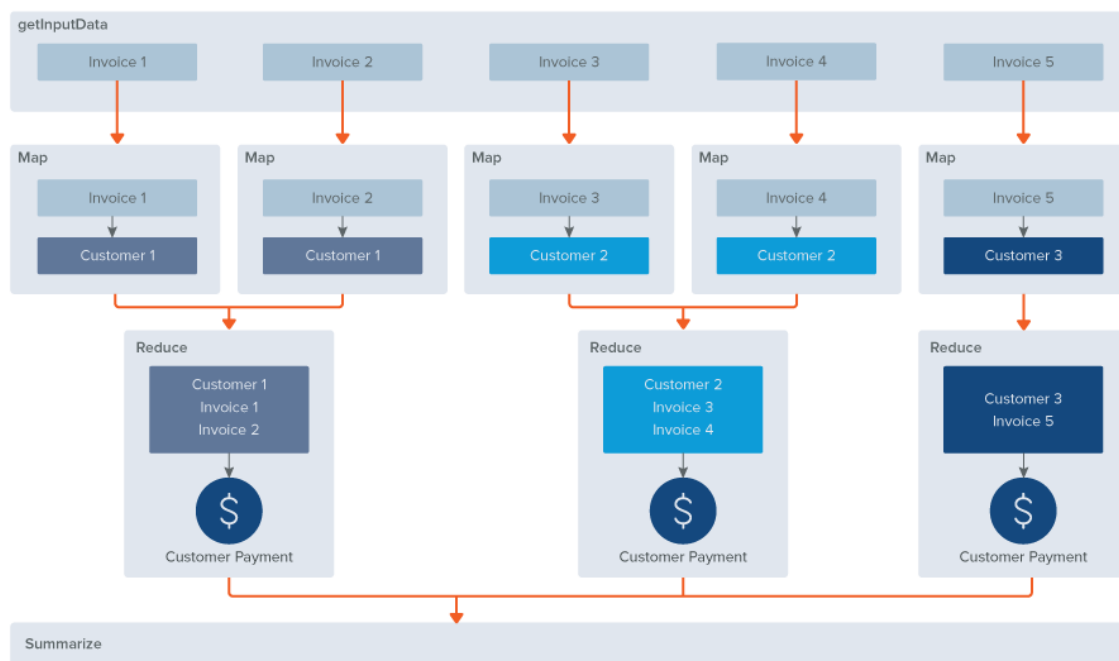
Note: It is not mandatory to use both the map stage and the reduce stage. You may skip one of those stages.

The following diagram illustrates these stages, in the context of processing a set of invoices.

In this example, the stages are used as follows:

- **Get Input Data** – A collection of invoices that require payment is loaded.
- **Map** – Each invoice is paired to the customer expected to pay it. The key/value pairs are returned, where customerID is the key and the invoice is the value. For five invoices, the map is invoked five times.
- **Reduce** – There are three unique groups of invoices based on customerID. For three groups, reduce is invoked three times. To create a customer payment for every group, custom logic iterates over each group using customerID as the key.
- **Summarize** – Custom logic fetches various metrics (for example, number of invoices paid) and sends the output as an email notification.

For a code sample similar to this example, see [Example: Script that Processes Invoices](#).



Passing Data to a Map or Reduce Stage

To prevent unintended alteration of data when it is passed between stages, key/value pairs are always serialized into strings. For map/reduce scripts, SuiteScript 2.0 checks if the data passed to the next stage is a string, and uses `JSON.stringify()` to convert the key or value into a string as necessary.

Objects serialized to JSON remain in JSON format. To avoid possible errors, SuiteScript does not automatically deserialize the data. For example, an error might result from an attempt to convert structured data types (such as CSV or XML) that are not valid JSON. At your discretion, you can use `JSON.parse()` to convert the JSON string back into a native JS object.

SuiteScript 2.0 Map/Reduce Yielding

A key advantage of the map/reduce script type is that it can manage some aspects of its own resource consumption. This behavior is achieved through a feature known as yielding.

To understand yielding, first be aware that all map/reduce scripts are processed by [SuiteCloud Processors](#). A processor is a virtual unit of processing power that executes a job.

With yielding, the system waits until after the completion of each map or reduce function invocation, then checks to see whether the job has exceeded certain limits. If it has, job gracefully ends its execution, making it possible for other jobs to gain access to the processor. A new job is created to take the place of the map/reduce job that ended. The new job has the same priority as the old one, but a later timestamp.

Yielding can occur after the following limits are surpassed:

- The time limit specified by the Yield After Minutes field on the script deployment record. You can set this value to any number between 3 and 60. The default is 60. For more details on this field, see [Yield After Minutes](#).
- The governance limit of 10,000 usage units per map or reduce job. For more details, see [Soft Limits on Long-Running Map and Reduce Jobs](#).

Because the system checks these limits only between function invocations, yielding never interrupts a function invocation. Additionally, be aware that a job does not yield until it has actually passed one of the relevant limits.

Yielding is unrelated to the governance limits that exist for a single invocation of a map or reduce function. Those limits are described in [Hard Limits on Function Invocations](#). Exceeding the limit for a single invocation of a map or reduce function causes the system to throw an SSS_USAGE_LIMIT_EXCEEDED error and ends the function invocation, even if it is not complete.

For help understanding how the map and reduce stages can each have multiple function invocations, review [Map/Reduce Key Concepts](#).



Important: Map/reduce yielding is automatic. There is no API for manually forcing a map/reduce job to yield. This behavior differs from the functionality that was available for SuiteScript 1.0 scheduled scripts.

SuiteScript 2.0 Map/Reduce Script Submission

You can submit a map/reduce script for processing in any of the following ways:

- By [Scheduling a Map/Reduce Script Submission](#)
- By [Submitting an On-Demand Map/Reduce Script Deployment from the UI](#)
- By [Submitting an On-Demand Map/Reduce Script Deployment from a Script](#)

Note that you can create multiple deployments for one script record. This strategy is useful if you want to submit the same script for processing multiple times simultaneously, or within a short time. For details, see [Submitting Multiple Deployments of the Same Script](#).

Each of these processes requires you to use a map/reduce script deployment record. For details about the fields available on this record, see [Map/Reduce Script Deployment Record](#).



Important: All map/reduce script instances are executed, or processed, by SuiteCloud Processors. Before submitting a map/reduce script for processing, review [SuiteCloud Processors](#).

Map/Reduce Script Deployment Record

Before a map/reduce script can be executed, you must create at least one deployment record for the script.

The deployment record for a map/reduce script is similar to that of other script types. However, a map/reduce script deployment contains some additional fields. Some of these fields are specific to [SuiteCloud Processors](#), which are used to execute map/reduce scripts. Others are specific to map/reduce features. This topic describes all of the available fields.

You can access a map/reduce script deployment record in the following ways:

- To open an existing deployment record for editing, go to Customization > Scripting > Script Deployments. Locate the appropriate record, and click the corresponding **Edit** link.
- To start creating a new deployment record, open the appropriate script record in view mode, and click the **Deploy Script** button. For help creating a script record, see [Script Record Creation](#).

Body Fields

The following table summarizes the fields available on the map/reduce script deployment record. Note that some fields are available only when you edit or view an existing deployment record.

Field	Description
Script	A link to the script record associated with the deployment. This value cannot be changed, even on a new deployment record. If you begin the process of creating a deployment and realize that you selected the wrong script record, you must start the process over.
Title	The user-defined name for the deployment.
ID	<p>A unique identifier for the deployment.</p> <p>On a new record, you can customize this identifier by entering a value in the ID field. You should customize the ID if you plan to bundle the deployment for installation into another account, because using custom IDs helps avoid naming conflicts. IDs must be lowercase and cannot use spaces. You can use an underscore as a delineator.</p> <p>If you do not enter a value, the system automatically generates one. In both cases, the system automatically adds the prefix <code>customdeploy</code> to the ID created when the record is saved.</p> <p>Although not recommended, you can change the ID on an existing deployment by clicking the Change ID button.</p>
Deployed	<p>A configuration option that indicates whether the deployment is active. This box must be checked if you want the script to execute. Otherwise, the system uses the following behavior:</p> <ul style="list-style-type: none"> ■ When the Deployed box is cleared, and the Status is Not Scheduled, the Save and Execute option is no longer available, and the deployment cannot be submitted programmatically. ■ When the Deployed box is cleared, and the deployment record's Status is Scheduled, any times configured on the Schedule subtab are ignored and the script deployment is not submitted.
Status	<p>A value that determines how and when a script deployment can be submitted for processing. The primary options are:</p> <ul style="list-style-type: none"> ■ Scheduled —The script deployment is submitted for processing at the times indicated on the Schedule subtab.

Field	Description
	<ul style="list-style-type: none"> ■ Not Scheduled — The script deployment is submitted for processing only when invoked manually, either through the UI or programmatically. <p>Note also that, regardless of the Status, the system submits the deployment for processing only if the Deployed box is checked.</p> <p>For more details on this choice, see Status.</p>
See Instances	<p>A link to the Map/Reduce Script Status Page, filtered for all instances of this deployment record, for the current day. You can change the filtering options if needed.</p> <p>For details on working with the Map/Reduce Script Status page, see SuiteScript 2.0 Map/Reduce Script Status Page.</p>
Log Level	<p>A value that determines what type of log messages are displayed on the Execution Log of both the deployment record and associated script record. The available levels are:</p> <ul style="list-style-type: none"> ■ Debug — suitable for scripts still being tested; this level shows all debug, audit, error and emergency messages. ■ Use Audit, Error, or Emergency — suitable for scripts in production mode. Of these three, Audit is the most inclusive and Emergency the most exclusive. <p>For more details on each level, see Log Level.</p>
Execute As Role	<p>Indicates the role used to run the script. For map/reduce deployments, this value is automatically set to Administrator and cannot be edited.</p>
Priority	<p>A measure of how urgently this script should be processed relative to other map/reduce and scheduled scripts that have been submitted. This value is assigned to each job associated with the script deployment. The priority affects when the SuiteCloud Processors scheduler sends these jobs to the processor pool. For more details, see Priority.</p> <div style="border: 1px solid #f1c40f; padding: 10px; margin-top: 10px;"> <p> Important: You must understand SuiteCloud Processors before you change this setting. For details, see the help topic SuiteCloud Processors – Priority Levels.</p> </div>
Concurrency Limit	<p>Determines the number of SuiteCloud Processors that can be used to process the jobs associated with the script deployment. For more details on this field, see Concurrency Limit.</p>
Submit All Stages At Once	<p>Determines whether the system creates jobs for all of the map/reduce stages simultaneously when the script deployment is submitted for processing. In general, clear this field only if the script deployment is relatively low in priority. For more details on this field, see Submit All Stages At Once.</p>
Yield After Minutes	<p>A soft time limit on how long the script deployment's map and reduce jobs may run before yielding. You can enter a number from 3 to 60. In general, each time a map or reduce job finishes a function invocation, the system checks to see whether this time limit has been exceeded. If it has, the job yields so that other jobs can be processed. A new job is created to take on the work that was being processed by the job that yielded. For more details on this field, see Yield After Minutes.</p>
Buffer Size	<p>A value that indicates how many key/value pairs a map or reduce job can process before information about the job's progress is saved to the database. A low Buffer Size value minimizes the risk of any pairs being processed twice, which can lead to data duplication. In general, leave this value set to 1 unless you have special circumstances that dictate otherwise. For more details on this field, see Buffer Size.</p>

Status

The Status field determines how and when the script deployment may be submitted for processing. The default value is Not Scheduled.

Regardless of how the script deployment is submitted, it **does not necessarily execute** at the exact time scheduled, or at the exact time that it is manually invoked. There may be a short system delay, even if no scripts are before it. If there are scripts already waiting to be executed, the script may

need to wait until others have completed. For details on this behavior, see the help topic [SuiteCloud Processors](#).

Scheduled

When a deployment's Status is set to **Scheduled**, the script deployment is submitted for processing according to a one-time or recurring schedule. You define this schedule by using the deployment record's [Schedule Subtab](#). With this approach, after you save the deployment, you do not have to take any other steps for the deployment to be submitted for processing.

Note also:

- If you schedule a recurring submission with an end date, or a one-time submission, the deployment record's status remains Scheduled even after the script completes its execution.
- You cannot submit an on-demand instance of the script deployment when it has a status of Scheduled.

See also [Scheduling a Map/Reduce Script Submission](#).

Not Scheduled

When a deployment's Status is set to **Not Scheduled**, the deployment is available to be submitted on an on-demand basis. If you want the deployment to be submitted for processing, you must manually submit it, either through the NetSuite UI or programmatically. You can use either of the following:

- The **Save and Execute** option on the deployment record. See also [Submitting an On-Demand Map/Reduce Script Deployment from the UI](#).
- The `task.ScheduledScriptTask` API. See also [Submitting an On-Demand Map/Reduce Script Deployment from a Script](#).

You can submit the script deployment only if there is no other unfinished instance of the same script deployment. If you want multiple instances of the script to be submitted for processing simultaneously, you must create multiple deployment records for the script record. For details, see [Submitting Multiple Deployments of the Same Script](#).

Testing

When a deployment's Status is set to **Not Scheduled**, the script deployment is submitted for processing only when invoked manually, either through the UI or programmatically. You can submit the script deployment only if there is no other unfinished instance of the same script deployment.

To be submitted programmatically, a deployment with a status of Testing must be explicitly identified by the `task.MapReduceScriptTask` object's `MapReduceScriptTask.deploymentId` property.

Log Level

The Log Level field determines what type of log messages are displayed in the Execution Log.

In general, if a script is still being tested, use the **Debug** log level. This option includes more messages than the other log levels, including messages created by `log.debug(options)`, `log.audit(options)`, `log.error(options)`, and `log.emergency(options)`.

If a script is in production, use one of the following levels:

- **Audit** — This level includes a record of events that have occurred during the processing of the script (for example, "A request was made to an external site"). This level includes log messages created by `log.audit(options)`, `log.error(options)`, and `log.emergency(options)`.
- **Error** — A log level set to Error shows only unexpected script errors, including log messages created by `log.error(options)` and `log.emergency(options)`.
- **Emergency** — Includes only the most critical messages, including log messages created by `log.emergency(options)`.

The default value is Debug.

Priority

If multiple map/reduce and scheduled script deployments are submitted for processing at the same time, some scripts may have to wait to be processed. To handle this type of situation, you can use the **Priority** field. This setting determines how quickly the script deployment instance, and each of its jobs, should be sent for processing relative to other script deployment instances that were created at the same time. The priority for the deployment is applied to each of the deployment instance's jobs.

The choices are as follows:

- **High** — Use to mark critical deployments that require more immediate processing. The scheduler sends high-priority jobs to the processor pool first.
- **Standard** — This is the default setting. It is considered to be a medium priority level. The scheduler sends medium-priority jobs to the processor pool if there are no high-priority jobs waiting.
- **Low** — Use to mark deployments that can tolerate a longer wait time. The scheduler sends low-priority jobs to the processor pool if there are no high- or standard-priority jobs waiting.



Important: You must understand [SuiteCloud Processors](#) before you change this setting. See the help topic [SuiteCloud Processors – Priority Levels](#).

Concurrency Limit

The map/reduce script type permits parallel processing. With parallel processing, multiple [SuiteCloud Processors](#) can work simultaneously to execute a single script deployment instance. You can control the number of processors used for each script instance through the deployment record's **Concurrency Limit** field.

Note that this setting affects the map and reduce stages only. (These stages are the only ones that permit parallel processing.)

For example, if you specify a Concurrency Limit of 5, the system creates five map jobs and five reduce jobs. If you leave this field set to no number, then the maximum number of processors available to your account is used. The default value is 1.



Note: The Concurrency Limit field was introduced in 2017.2, as part of the SuiteCloud Processors feature. If you are editing a deployment record that was created prior to 2017.2, be aware that when your account was upgraded, the Concurrency Value field was initially set to a value that corresponds to the number of queues that had been saved for the Queues field.

Submit All Stages At Once

Every map/reduce script deployment instance is processed by multiple jobs. At least one job is created for each stage used by the script. Every map/reduce script must use either four or five stages: getInputData, shuffle, summarize, and either map or reduce (or both). However, the jobs for the various stages are not necessarily submitted at the same time. This behavior is controlled by the **Submit All Stages at Once** option.

Map/reduce stages must occur in a specific sequence. When the Submit All Stages at Once option is disabled, the system waits to submit the jobs for each stage until after the prerequisite job completes.

In contrast, when the Submit All Stages at Once option is enabled, the system submits jobs for all stages simultaneously. This behavior increases the likelihood that all jobs associated with the script deployment instance finish, without gaps, before another script begins executing. However, be aware that this option **does not guarantee** that no gaps occur. For example, because a map/reduce job can yield, a long-running job may be forced to end, and a job associated with another script may begin executing in its place. For these reasons, you should not rely on this option if you need to enforce a strict execution sequence among scripts. The only way to enforce a strict sequence is to have one script

schedule another during the summarize stage. A script can be scheduled programmatically by using the `task.create(options)` method. For more details, see [Submitting an On-Demand Map/Reduce Script Deployment from a Script](#).

The Submit All Stages at Once option is enabled by default. In general, Oracle recommends that you leave this option enabled.

Yield After Minutes

The Yield After Minutes field helps you prevent any processor from being monopolized by a long-running map or reduce job.

Here's how this setting works: During the map and reduce stages, after each function invocation, the system checks to see how long the map or reduce job has been running. If the amount of elapsed time has surpassed the number of minutes identified in the Yield After Minutes field, the job gracefully ends its execution, and a new job is created to take its place. The new job has the same priority as the old one, but a later timestamp. This behavior is known as yielding.

By default, Yield After Minutes is set to 60, but you can enter any number from 3 to 60.

The system never interrupts a function invocation for this limit. Also, the system never ends a map or reduce job before the limit has been reached, but only after it has been passed. For that reason, the degree to which the limit is surpassed varies depending on the duration of your function invocation. For example, if the Yield After Minutes limit is 3 minutes, but your function takes 15 minutes to complete, then in practice the job yields after 15 minutes, not 3 minutes.

Yielding is also affected by a governance limit. This limit is 10,000 usage units for each map and reduce job. This limit works in the same way as the Yield After Minutes limit: The system waits until after each function invocation ends to determine whether the usage-unit limit has been surpassed. If it has, the job yields, even if the Yield After Minutes limit has not been exceeded.

See also [SuiteScript 2.0 Map/Reduce Yielding](#) and [Soft Limits on Long-Running Map and Reduce Jobs](#).

Buffer Size

The purpose of the **Buffer Size** field is to control how frequently a map or reduce job saves data about its progress. In general, Oracle recommends that you leave this field set to 1.

To understand this setting, remember how map and reduce jobs work: First, the job flags some number of key/value pairs that require processing. Then it processes the pairs that it flagged. Then it saves data about the work it did. This data includes information about which key/value pairs were processed, how many usage points were consumed, and so on. The process repeats either until the job yields or until no key/value pairs remain.

The Buffer Size field controls how many pairs are flagged at one time. So if you leave this field set to its default of 1, the job flags one pair, processes it, and saves the data. Then it repeats the cycle.

You can set Buffer Size to any of the following values: 1, 2, 4, 8, 16, 32, or 64. However, the disadvantage of choosing a higher number is that, if the job is interrupted by an application server restart, there is a greater likelihood that one or more key/value pairs will be processed twice. On the other hand, setting this field to a higher number can be more efficient in certain cases.

Use the following guidance:

- In general, leave this value set to 1, particularly if the script is processing records.
- You can choose a higher buffer size value if the script is performing fast, algorithmic operations, or if other special circumstances dictate that you deviate from the default setting.

Schedule Subtab

The following table summarizes the fields on the Schedule subtab. These settings are honored only if the deployment record's **Status** is set to Scheduled and the Deployed box is checked.

Field	Description
Single Event	The map/reduce script deployment is submitted only once. Use this option if you want to schedule a future one-time submission.
Daily Event	The map/reduce script deployment is submitted every x number of days. If you schedule the submission to recur every x minutes or hours, the schedule starts over on the next scheduled day. For example, your deployment is set to submit daily, starting at 3:00 am and recurring every five hours. A scheduled script instance is submitted at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. At midnight, the schedule starts over and the next submission is at 3:00 am.
Weekly Event	The map/reduce script deployment is submitted at least one time per scheduled week. If you schedule the submission to recur every x minutes or hours, the schedule starts over on the next scheduled day. For example, your deployment is set to submit on Tuesday and Wednesday, starting at 3:00 am and recurring every five hours. The deployment is submitted on Tuesday at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. On Wednesday, the schedule starts over and the next submission is at 3:00 am.
Monthly Event	The map/reduce script deployment is submitted at least one time per month.
Yearly Event	The scheduled script deployment is submitted at least one time per year.
Start Date	The first submission occurs on this date. This field is mandatory if a one-time or recurring schedule is set.
Start Time	If a value is selected, the first submission occurs at this time.
Repeat	If a value is selected, the first submission occurs on the date and time selected. A new script deployment instance is then created and submitted every x minutes or hours until the end of the start date. If applicable, the schedule starts over on the next scheduled day. For example, your deployment is set to submit on Tuesday and Wednesday, starting at 3:00 am and recurring every five hours. The deployment is submitted on Tuesday at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. On Wednesday, the schedule starts over and the next submission is at 3:00 am.
End By	If a value is entered, the last submission ends by this date. If you schedule the submission to recur every x minutes or hours, a new script deployment instance is created and submitted every x minutes or hours until the end date.
No End Date	The schedule does not have a set end date.

Scheduling a Map/Reduce Script Submission

Map/reduce script deployments can be submitted for processing on a scheduled basis. For example, you could configure the following schedules:

- A one-time submission, at a predefined time.
- Repeated submission, on a daily, weekly, monthly, or yearly basis.

To set a scheduled submission, the Status field on the deployment record must be set to **Scheduled**. Additionally, you must configure one or more upcoming times on the record's Schedule subtab.

Deployment times can be scheduled with a frequency of every 15 minutes. For example, you could configure a script to run at 2:00 pm, 2:15 pm, 2:30 pm, and so on.

The times you set on the Schedule subtab are the times the script deployment is submitted for processing. However, **the times you set on the Schedule subtab are not necessarily the times the script will execute**. Script deployment does not mean the script will actually execute precisely at 2:00

pm, 2:15 pm, 2:30 pm, and so on. There may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may wait to be executed until other scripts have completed. For more details about how map/reduce scripts are processed, see the help topic [SuiteCloud Processors](#).

To schedule a one-time or recurring map/reduce script submission:

1. Create your map/reduce script entry point script. This process includes uploading a JavaScript file and creating a script record based on that file. To review a map/reduce script example, see [SuiteScript 2.0 Map/Reduce Script Type](#). If this script is your first, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. Open the appropriate script record in view mode. Click the **Deploy Script** button.
3. When the script deployment record loads, check the **Deployed** box, if it is not already checked.
4. Set the **Status** field to **Scheduled**.
5. Set the remaining body fields. For help understanding the fields, see [Map/Reduce Script Deployment Record](#).
6. On the **Schedule Subtab**, set the deployment options.

For example, if you wanted to **submit the script hourly**, you would configure the subtab as follows:

- Deployed = checked
- Daily Event = [radio button enabled]
- Repeat every 1 day
- Start Date = [today's date]
- **Start Time = 12:00 am**
- Repeat = every hour
- End By = [blank]
- No End Date = checked
- Status = Scheduled
- Log Level = Error

If the **Start Time** is set to any other time than 12:00 am (for example, it is set to 2:00 pm), the script will start at 2:00 pm, but then finish its hourly execution at 12:00 am. It will not resume until the next day at 2:00 pm.

7. Click **Save**.

Note: In some cases, you may want to submit a map/reduce script for processing multiple times simultaneously, or within a short time frame. However, the system does not permit a script deployment to be submitted if a previous instance of the deployment has already been submitted and is not yet finished. The solution in this case is to create multiple deployments for the script. In other words, repeat Steps 2 through 7 of the procedure above as needed.

Submitting an On-Demand Map/Reduce Script Deployment from the UI

Map/reduce scripts can be submitted for processing on an on-demand basis from the NetSuite UI. To submit a script in this way, use the **Save and Execute** command on the Script Deployment page. The **Status** field on the deployment must be set to **Not Scheduled**.

You can also submit an on-demand deployment programmatically. For details, see [Submitting an On-Demand Map/Reduce Script Deployment from a Script](#).

Note that after you submit an on-demand deployment of a script, the script **does not necessarily execute right away**. After a script is submitted for processing, there may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may wait to be executed until other scripts have completed. For more details about how map/reduce scripts are processed, see the help topic [SuiteCloud Processors](#).

To submit an on-demand map/reduce script for processing from the UI:

1. Create your map/reduce script entry point script. This process includes uploading a JavaScript file and creating a script record based on that file. To review a map/reduce script example, see [SuiteScript 2.0 Map/Reduce Script Type](#). If this script is your first, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. Open the appropriate script record in view mode. Click the **Deploy Script** button.
3. When the script deployment record loads, check the **Deployed** box, if it is not already checked.
4. Set the Status field to **Not Scheduled**.
5. Set the remaining body fields. For help understanding the fields, see [Map/Reduce Script Deployment Record](#).
6. Select **Save and Execute** from the **Save** dropdown list.



Note: In some cases, you may want to submit a map/reduce script for processing multiple times simultaneously, or within a short time frame. However, the system does not permit a script deployment to be submitted if a previous instance of the deployment has already been submitted and is not yet finished. The solution in this case is to create multiple deployments for the script. In other words, repeat Steps 2 through 6 of the procedure above as needed.

Submitting an On-Demand Map/Reduce Script Deployment from a Script

Map/reduce scripts can be submitted for processing on an on-demand basis from another server-side script. You can submit a deployment this way by using the [task.ScheduledScriptTask](#) API. For the call to be successful, the **Status** field on the script deployment record must be set to **Not Scheduled**.

You can also submit an on-demand deployment from the UI. For details, see [Submitting an On-Demand Map/Reduce Script Deployment from the UI](#).

Note that after you submit an on-demand deployment of a script, the script **does not necessarily execute right away**. After a script is submitted for processing, there may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may wait to be executed until other scripts have completed. For more details about how map/reduce scripts are processed, see the help topic [SuiteCloud Processors](#).

To submit an on-demand map/reduce script instance from a script:

1. Create your map/reduce script entry point script. This process includes uploading a JavaScript file and creating a script record based on that file. To review a map/reduce script example, see [SuiteScript 2.0 Map/Reduce Script Type](#). If this script is your first, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. Open the appropriate script record in view mode. Click the **Deploy Script** button.
3. When the script deployment record loads, check the **Deployed** box, if it is not already checked.
4. Set the Status field to **Not Scheduled**.
5. Set the remaining body fields. For help understanding the fields, see [Map/Reduce Script Deployment Record](#).
6. Click **Save**.

7. In the server-side script where you want to submit the map/reduce script, call `task.create(options)` to return a `task.MapReduceScriptTask` object:

```
var scriptTask = task.create({taskType: task.TaskType.MAP_REDUCE});
```

8. Set the `MapReduceScriptTask.scriptId` and `MapReduceScriptTask.deploymentId` properties:

```
scriptTask.scriptId = 'customscript1';
scriptTask.deploymentId = 'customdeploy1';
```

Note that the system does not require you to set a value for the `deploymentId` property. If you omit this value, the system searches for and uses any available deployment record for the script record you identified with the `scriptId` property. For a deployment to be considered available, three conditions must be met: The deployment record must have a status of Not Scheduled, its Deployed option must be set to true, and no unfinished instances of the deployment can exist.

9. Call `MapReduceScriptTask.submit()` to submit the script for processing. For example:

```
scriptTask.submit();
```

Note: In some cases, you may want to submit a map/reduce script for processing multiple times simultaneously, or within a short time frame. However, the system does not permit a script deployment to be submitted if a previous instance of the deployment has already been submitted and is not yet finished. The solution in this case is to create multiple deployments for the script. In other words, repeat Steps 2 through 9 of the procedure above as needed. If you are using this approach, consider omitting the `deploymentId` property when you create the map/reduce script task object. When you omit the `deploymentId`, the system searches for and uses whichever deployment record is available.

Submitting Multiple Deployments of the Same Script

The system does not permit you to submit a single script deployment for processing multiple times simultaneously. That is, if one instance of the deployment has been submitted and not yet finished, you cannot submit the same deployment again. You must wait for the unfinished instance to complete.

If you need to submit multiple instances of a single map/reduce script for processing at the same time, or within a very short time frame, the correct approach is to create multiple deployments of the script and submit each deployment as needed.

There are multiple advantages to this technique. For example, this approach lets you process two or more instances of the script in parallel. Additionally, if you need a map/reduce script to be processed multiple times in the same general time frame, you can submit both instances at the same time without having to worry about the second submission failing with an error. The alternative would be to monitor the progress of the first script deployment instance and then be careful to submit the second deployment only after completion of the first.

Note that if a deployment has already been submitted but not yet finished, and you open the deployment record for editing in the UI, the deployment record's Save and Execute option is not available. If you try to submit the deployment programmatically in this case, the system throws the `MAP_REDUCE_ALREADY_RUNNING` error.

For help creating or submitting a deployment, see the following topics:

- [Scheduling a Map/Reduce Script Submission](#)
- [Submitting an On-Demand Map/Reduce Script Deployment from the UI](#)
- [Submitting an On-Demand Map/Reduce Script Deployment from a Script](#)

SuiteScript 2.0 Map/Reduce Governance

As with all script types, NetSuite imposes usage limits on map/reduce scripts. Governance rules for map/reduce scripts fall into two main categories:

- Certain limits, if violated, cause an interruption to the current function invocation. These limits are known as **hard limits**.
- Other limits are managed automatically by the system. These limits never interrupt a function invocation. Rather, after the completion of a function invocation, these limits can cause a job to yield and its work to be rescheduled for later. These limits are known as **soft limits**.

See the following sections for more details:

- [Hard Limits on Total Persisted Data](#)
- [Hard Limits on Function Invocations](#)
- [Soft Limits on Long-Running Map and Reduce Jobs](#)

Be aware that the system does not impose any limit on the full duration of a map/reduce script deployment instance. It also is not possible for a user to impose such a limit. Rather, the system's limits regulate isolated components of the deployment, such as the duration of a single function invocation.

Note: One way that NetSuite measures a script's activity is through usage units. For more information about usage units, see the help topic [SuiteScript Governance](#).

Hard Limits on Total Persisted Data

The total persisted data used by a map/reduce script cannot exceed 200MB at any one time. If your script exceeds this limit at any point during its processing, the system throws a `STORAGE_SIZE_EXCEEDED` error. Additionally, the script ends its current function invocation, exits the current stage, and goes immediately to the summarize stage. (This error does not occur in the summarize stage, because the total persisted data cannot be increased during that stage.)

Note the following about how persisted data is measured:

- The system takes into account any search results retrieved and returned by the input function. Note that a large number of columns in a result set can significantly increase the amount of data used.
- During the map and reduce stages, the total size is a measure of the keys and values yet to be processed. After a key or value is processed, it does not count toward the limit.

Hard Limits on Function Invocations

The following table describes the limits applied to a map/reduce script's function invocations. If your script exceeds any of these limits, the system throws an `SSS_USAGE_LIMIT_EXCEEDED` error. The way the system responds to this error varies depending on the stage and the configuration of your script, as shown in the following table.

Stage	Usage limit per function invocation	Response to <code>SSS_USAGE_LIMIT_EXCEEDED</code> error
Get Input Data	10,000 units	The script ends the function invocation and exits the stage. It proceeds directly to the summarize stage.
Map	1,000 units	The response includes two parts. Note that you can configure the second part: 1. The function invocation ends, even if its work on the current key/value pair is incomplete.
Reduce	5,000 units	

Stage	Usage limit per function invocation	Response to SSS_USAGE_LIMIT_EXCEEDED error
		2. The default behavior is that the job continues working and moves on to other key/value pairs. However, you can configure the script to invoke the function again for the same key/value pair. For details, see Configuration Options for Handling Map/Reduce Interruptions .
Summarize	10,000 units	The script stops executing.

Note that if you are using the map/reduce script type as intended, your script should not approach these limits, particularly for map and reduce function invocations. In general, each invocation of a map or reduce function should do a relatively small portion of work. For more details, see [SuiteScript 2.0 Map/Reduce Script Best Practices](#).

Soft Limits on Long-Running Map and Reduce Jobs

In addition to the limits described in [Hard Limits on Function Invocations](#), the system includes a soft limit of 10,000 usage units on each map and reduce job.

To understand how this limit works, first be aware that all map/reduce scripts are processed by [SuiteCloud Processors](#). A processor is a virtual unit of processing power that executes a job.

The 10,000-unit soft limit is a mechanism designed to prevent any processor from being monopolized by a long-running map or reduce job. During the map and reduce stages, after each function invocation, the system checks the total number of units that have been used by the job. If the total usage has surpassed 10,000 units, the job gracefully ends its execution and a new job is created to take its place. The new job has the same priority as the old one, but a later timestamp. This behavior is known as yielding.

Yielding is also affected by the script deployment record's Yield After Minutes field. This time limit works in the same way as the 10,000-unit limit: The system waits until after each function invocation ends to determine whether the time limit has been exceeded. If it has, the job yields, even if the 10,000-unit limit has not been exceeded. By default, Yield After Minutes is set to 60, but you can enter any number from 3 to 60. For more details, see [Yield After Minutes](#). See also [SuiteScript 2.0 Map/Reduce Yielding](#).



Important: Yielding is unrelated to the limits that exist for a single invocation of a map function and a single invocation of a reduce function. Those limits are described in [Hard Limits on Function Invocations](#). Exceeding the limits for a single invocation of a map or reduce function causes the system to throw an SSS_USAGE_LIMIT_EXCEEDED error and ends the function invocation, even if it is not complete.

SuiteScript 2.0 Map/Reduce Script Status Page

- [Viewing Map/Reduce Script Status](#)
- [Viewing Details of Script Instances](#)
- [Programmatically Retrieving Script Instance Details](#)

Viewing Map/Reduce Script Status

You can monitor map/reduce script execution via the map/reduce script status page in the UI. With the script status page, you can see whether a map/reduce script deployment is pending, in progress, or unable to complete.

If all tasks are pending, you can cancel a script deployment from this page.

From NetSuite, go to Customization > Scripting > Map/Reduce Script Status.

To help you understand and optimize the performance of script entry points used, you can drill down for more details about map stages, processing utilization, and timing. You can use this information to gain insight about the time required to complete a stage or process a task.

Additionally, from the script status page, you can view the deployment record and consider changing the concurrency limit.

Viewing Details of Script Instances

For scheduled scripts and map/reduce scripts, a script instance is a scheduled script task or map/reduce script task that is submitted for processing. This script instance can also be called a task.

To view the details of a map/reduce script instance, from the Map/Reduce Script Status page, click **View Details**. Each row contains information about an individual map/reduce job that belongs to the script instance. A job always belongs to one of the stages of a map/reduce script, and there is at least one job per stage. There is exactly one job for each of the `getInputData`, `shuffle`, and `summarize` stages. The number of jobs for the map and reduce stages depends on concurrency limits and the number of yields.

For more information, see [SuiteScript 2.0 Map/Reduce Script Stages](#) and [SuiteScript 2.0 Map/Reduce Yielding](#).

Programmatically Retrieving Script Instance Details

To get the script status via the [N/search Module](#), create and load a search using `scheduledscriptinstance` as the type argument.

To get the status via the [N/task Module](#), see `task.MapReduceScriptTask`.

SuiteScript 2.0 Map/Reduce Script Testing and Troubleshooting

You have several options to test and debug your map/reduce script:

- Make a separate unit test suite as you develop and modify your map/reduce script. See [Map/Reduce Script Unit Testing](#).
- Check the execution logs. See [N/log Module](#).
- Monitor the script status. See [SuiteScript 2.0 Map/Reduce Script Status Page](#) and [Programmatically Retrieving Script Instance Details](#).
- Detect any server restarts that interrupted map/reduce script execution. See [inputContext.isRestarted](#), [mapContext.isRestarted](#), [reduceContext.isRestarted](#), and [summaryContext.isRestarted](#).

Do not use the SuiteScript Debugger for deployed debugging of a map/reduce script type. However, you may wish to test any dependencies on other types of scripts. Remember that to test existing scripts, the Script Deployment Status must be set to Testing and the currently logged in user must be listed as the script record owner. For information about the SuiteScript Debugger, see the help topic [SuiteScript Debugger](#).

Map/Reduce Script Unit Testing

 **Note:** You cannot use the SuiteScript Debugger for server-side script debugging of a Map/Reduce script.

To test a map/reduce script on demand, NetSuite recommends splitting the script into entry point level sections. Use the sections to form unit tests. Each section should function as an entry point script that

can be executed without external dependencies. If passing in modules, make sure that you use the `require` function and absolute paths.

To test map or reduce stages, you will need to create a mock context that seeds values and provides the dependent objects and parameters. Then, check the states, behavior, inputs, and outputs of map/reduce functions using assertion statements. Note that the `getInputData` stage does not take parameters, so it will not require mock context and can be tested more conventionally.

To test a summarize stage, if it contains logic operating on a final set of data that has no return, use assertions and logs to gather information.

To assist development of your unit test, download the SuiteScript 2.0 API files, specifically those representing the `mapContext`, `reduceContext`, and `summaryContext` objects. These file can act as a schema for the properties and methods you want to test or mock. To access the files, do the following:

1. From NetSuite, select Documents > Files > File Cabinet.
2. Select SuiteScript 2.0 API to download the zip folder.
3. Extract the `mapReduceContext` and `mapReduceSummary.js` files.

For an example of unit testing for SuiteScript 2.0 scripts, see the help topic [Project Sample: SuiteScript 2.0 Unit Testing](#).

SuiteScript 2.0 Map/Reduce Script Error Handling

A map/reduce script can be interrupted by either of the following:

- An application-server disruption, which can occur because of a NetSuite update, NetSuite maintenance, or an unexpected failure of the execution environment
- An uncaught error

Interruptions can leave portions of a script's work incomplete. However, the system includes measures for dealing with this problem.

After an **application-server disruption**, the system automatically restarts the script, regardless of which stage was in progress when the failure occurred. For example, if the failure occurred during the map or reduce stage, the map or reduce job restarts. In this case, the default behavior is that the restarted job invokes the map or reduce function again for all key/value pairs that were previously flagged for processing but not yet marked complete. This behavior can be modified by using the `retryCount` option.

After an **uncaught error** is thrown, the behavior varies depending on the stage. If the error is thrown during the `getInputData` stage, the script goes immediately to the summarize stage, and the `getInputData` stage is not restarted. If an uncaught error is thrown during the map or reduce stage, the default behavior is that the current function invocation ends, even if incomplete, and the map or reduce job moves on to other key/value pairs that require processing. However, the script can be configured so that the map or reduce function is invoked again for the pair that was being processed when the error occurred. You manage the system's behavior in this case by using the `retryCount` and `exitOnError` options.

To fully understand the system response to interruptions, see [System Response After a Map/Reduce Interruption](#).

In general, your script should include logic that checks to see whether a restart has occurred. If the function has been restarted, the script should take any actions needed to avoid unwanted duplicate processing. For more information, see [Adding Logic to Handle Map/Reduce Restarts](#).

For additional explanation of how each stage responds to a restart, see [Execution of Restarted Map/Reduce Stages](#).

System Response After a Map/Reduce Interruption

A map/reduce script can be interrupted at any time. For example, a disruption to the application server immediately stops the script's execution. Additionally, an uncaught error, although it does not cause the script to stop executing, stops the current function invocation, even if it is not complete.

For more details, review the following sections:

- [System Response After an Application-Server Disruption](#)
- [System Response After an Uncaught Error](#)



Important: Regardless of how your script is configured, you should make sure that it includes logic that checks to see whether a restart has occurred. If the function has been restarted, the script should take any actions needed to avoid unwanted duplicate processing. For details, see [Adding Logic to Handle Map/Reduce Restarts](#).

System Response After an Application-Server Disruption

An application disruption can occur because of a NetSuite update, NetSuite maintenance, or an unexpected failure of the execution environment. When the application server is disrupted in this way, the script stops executing. After the application server restarts, the script also restarts, resuming the same stage that it was in process when the script was interrupted.

When an application server restart interrupts the map or reduce stage, the system writes the `SSS_APP_SERVER_RESTART` error code to the relevant iterators. This error code is shown alongside the codes recorded for any uncaught errors that were thrown.

For more details, see the following table.

Stage where interruption occurred	Script behavior	SSS_APP_SERVER_RESTART error code written to
Get Input Data	When the disruption occurs, the script stops executing. After the application server restarts, the system restarts the function.	n/a
Map	When the disruption occurs, the script stops executing. Any data that was saved during the previous invocation by using the <code>context.write()</code> method is discarded. Afterward, the response is as follows:	<ul style="list-style-type: none"> ■ mapContext.errors — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ mapSummary.errors — Contains all error codes recorded during the map stage.
Reduce	<ol style="list-style-type: none"> 1. The system evaluates the retryCount config setting. If <code>retryCount</code> is set to a value greater than 0 or if the <code>retryCount</code> setting is not used, the script tries to process the same set of key/value pairs it was processing when the application server became unavailable. This data includes all pairs that were flagged for processing but not marked complete. However, if <code>retryCount</code> is set to 0, the script moves on to Step 2 without attempting further processing for these key/value pairs. 2. The job moves on to other key/value pairs that require processing and were not previously flagged as in progress. 	<ul style="list-style-type: none"> ■ reduceContext.errors — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ reduceSummary.errors — Contains all error codes recorded during the reduce stage.

Stage where interruption occurred	Script behavior	SSS_APP_SERVER_RESTART error code written to
Summarize	When the disruption occurs, the entire script stops executing. After the application server restarts, the system restarts the function.	n/a

System Response After an Uncaught Error

An error that is not caught in a try-catch block does not necessarily end the execution of a map/reduce script, but the error can disrupt the script's work. Some aspects of this behavior are configurable. For details, see the following table.

Stage where error occurred	Script behavior	Errors written to
Get Input Data	The script ends the function invocation and exits the stage. It proceeds directly to the summarize stage. This behavior cannot be configured.	<code>inputSummary.error</code>
Map	When the error occurs, the function invocation ends, even if its work is not complete. Any data that was saved during the invocation by using the <code>context.write()</code> method is discarded. Afterward, the system responds as follows: <ol style="list-style-type: none"> 1. The system evaluates the <code>retryCount</code> setting. If <code>retryCount</code> is set to a value greater than 0, and the maximum number of retries has not yet been used, the script tries to process the same key/value pair again. The rest of the steps in this process are not used. 	<ul style="list-style-type: none"> ■ <code>mapContext.errors</code> — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ <code>mapSummary.errors</code> — Contains all error codes recorded during the map stage.
Reduce	<ol style="list-style-type: none"> 2. The system evaluates the <code>exitOnError</code> setting. If <code>exitOnError</code> is set to true, the script exits the current stage and proceeds directly to the summarize stage. The rest of the steps in this process are not used. 3. The job continues by moving on to other key/value pairs that require processing. It does not do any further work on the pair it was processing when the error occurred. 	<ul style="list-style-type: none"> ■ <code>reduceContext.errors</code> — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ <code>reduceSummary.errors</code> — Contains all error codes recorded during the reduce stage.
Summarize	The script stops executing. This behavior cannot be configured.	n/a



Note: This table describes the behavior for the majority of errors, but a few errors result in different behavior. For example, the `SSS_USAGE_LIMIT_EXCEEDED` error causes a script to exit the map or reduce stage immediately. For details, see [Hard Limits on Total Persisted Data](#).

Configuration Options for Handling Map/Reduce Interruptions

A map/reduce script can be interrupted at any time. For example, a disruption to the application server immediately stops the script's execution. Additionally, an uncaught error, although it does not cause the script to stop executing, stops the current function invocation, even if it is not complete.

The system includes two configuration options that let you fine-tune the overall response of your script to interruptions. For details, see the following sections:

- `retryCount`
- `exitOnError`
- [Adding a Configuration Option to a Script](#)

retryCount

The `retryCount` option affects the map and reduce stages only. This option lets you configure your script to restart the map or reduce function for any key/pairs that were left in an uncertain state following an interruption, including application server restarts and uncaught errors.

The effect of this setting varies slightly depending on whether the script was disrupted by application server restart or an uncaught error. For example:

- When an **application server restart** occurs, the script cannot identify the exact key/value pair that was being processed when the interruption occurred. However, the script can identify the pairs that had been flagged for processing but were not yet marked as complete. In the event of a retry, the script retries processing for all of these pairs.
- When an **uncaught error** occurs, the script can identify the exact key/value pair that was being processed when the interruption occurred. When the `retryCount` option is being used, the script invokes the map or reduce function again for that specific key/value pair.

Valid values are 0 to 3. The number you choose for the `retryCount` setting is the number of retries permitted for **each** key/value pair that was left in an uncertain state after an interruption. For example, suppose you have `retryCount` set to 2, and an error interrupts a map job. In this case, the script would restart the function for the key/value pair that was being processed when the error was thrown. If the second function invocation for that pair was also interrupted by an error, the script would invoke the function for that key/value pair another time. However, if an error was thrown during the third attempt, the script would not retry processing again, because the `retryCount` setting permits only two retries. Later, when the job starts processing a different key/value pair, two retries are again available.

The `retryCount` setting is optional. You can set a value for it in the return statement of your map/reduce script, as described in [Adding a Configuration Option to a Script](#).

If you do not add the `retryCount` option to your script, the behavior varies depending on whether the script was disrupted by an uncaught error or by an application server restart. For example:

- When `retryCount` is not used and an application server restart interrupts the script, the system **restarts** processing for all key/value pairs that were left in an uncertain state.
- When `retryCount` is not used and an error interrupts the map or reduce function, the system **does not restart** processing for the key/value pair that was left in an uncertain state.

See also [System Response After an Uncaught Error](#) and [System Response After an Application-Server Disruption](#).



Note: In the case of an uncaught error, the system's overall response is also affected by the value of the `exitOnError` option. However, the script evaluates and reacts to the `retryCount` setting first.

exitOnError

The `exitOnError` option is used when an uncaught error occurs in the map or reduce stage. It is evaluated after the `retryCount` option.

When `exitOnError` is set to true, the script exits the stage after both of the following occur:

- An error is thrown and not caught.
- All retries permitted by the [retryCount](#) option have been exhausted.

This setting has no impact on the system's behavior after an application server restart. It is used only when an uncaught error is thrown.

The `exitOnError` setting is optional. You can set a value for it by adding it in the return statement of your map/reduce script, as described in [Adding a Configuration Option to a Script](#). Possible values are:

- **false** — The script continues processing in the current stage. (This is also the behavior used when the option is not added to the script.)
- **true** — The script exits the stage and goes immediately to the summarize stage.

See also [System Response After an Uncaught Error](#).

Adding a Configuration Option to a Script

If you want to use the [retryCount](#) or [exitOnError](#) option, add them to the script's return statement in a config block. For example:

```
// Add additional code.
...

return {

  config:{
    retryCount: 3,
    exitOnError: true
  },

  getInputData: myGetInputData,
  map: myMap,
  reduce: myReduce,
  summarize: mySummarize

};

...
// Add additional code.
```

Logging Errors

With any map/reduce script, you should include logic that checks for errors that may have occurred during the `getInputData`, `map`, and `reduce` stages. You can access data about errors using the context objects made available to each entry point. Use the properties shown in the following table.

Stage where error occurred	Property that contains data
Get Input Data	<code>inputSummary.error</code>
Map	<ul style="list-style-type: none"> ■ <code>mapContext.errors</code> — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ <code>mapSummary.errors</code> — Contains all error codes recorded during the map stage.

Stage where error occurred	Property that contains data
Reduce	<ul style="list-style-type: none"> ■ <code>reduceContext.errors</code> — Contains the error codes recorded during previous attempts to process the current key/value pair. ■ <code>reduceSummary.errors</code> — Contains all error codes recorded during the reduce stage.

Syntax

The following snippets shows how you can capture data about errors in various stages.

Map Stage

You can use this snippet in a map function. The snippet logs data only if an error was thrown during a previous invocation of the map function for the same key/value pair currently being processed.

```
// Create a log entry showing each full serialized error thrown
// during previous attempts to process the current key/value pair.

mapContext.errors.iterator().each(function (key, error, executionNo){

    log.error({
        title: 'Map error for key: ' + key + ', execution no ' + executionNo,
        details: error
    });

    return true;
});
```

Reduce Stage

You can use this snippet in a reduce function. The snippet logs data only if an error was thrown during a previous invocation of the reduce function for the same key/value pair currently being processed.

```
// Create a log entry showing each full serialized error thrown
// during previous attempts to process the current key/value pair.

reduceContext.errors.iterator().each(function (key, error, executionNo){

    log.error({
        title: 'Reduce error for key: ' + key + ', execution no ' + executionNo,
        details: error
    });

    return true;
});
```

Summarize Stage

You can use these snippets in a summarize function. They log data about errors thrown during previous stages.

```
// If an error was thrown during the input stage, log the error.

if (summary.inputSummary.error)
```

```

{
    log.error({
        title: 'Input Error',
        details: summary.inputSummary.error)
    });

    // For each error thrown during the map stage, log the error, the corresponding key,
    // and the execution number. The execution number indicates whether the error was
    // thrown during the the first attempt to process the key, or during a
    // subsequent attempt.

    summary.mapSummary.errors.iterator().each(function (key, error, executionNo){
        log.error({
            title: 'Map error for key: ' + key + ", execution no. " + executionNo,
            details: error
        });
        return true;
    });

    // For each error thrown during the reduce stage, log the error, the corresponding
    // key, and the execution number. The execution number indicates whether the error was
    // thrown during the the first attempt to process the key, or during a
    // subsequent attempt.

    summary.reduceSummary.errors.iterator().each(function (key, error, executionNo){
        log.error({
            title: 'Reduce error for key: ' + key + ', execution no. ' + executionNo,
            details: error
        });
        return true;
    });
}

```

Execution of Restarted Map/Reduce Stages

A map/reduce script can involve many jobs. The input, shuffle, and summarize stages are each processed with a single job. However, multiple jobs can participate in the map and reduce stages. Within a map stage or a reduce stage, jobs can run in parallel. Any of the jobs can be forcefully terminated at any moment. The impact of this event depends on the status of the job (what it was doing), and in which stage it was running.

For details, see the following topics:

- [Termination of getInput Stage](#)
- [Termination of Shuffle Stage](#)
- [Termination of Parallel Stages](#)
- [Termination of Summarize Stage](#)

Termination of getInput Stage

The work of a serial stage (getInput, shuffle, and summarize stages) is done in a single job. If the getInput stage job is forcefully terminated, it is later restarted. The getInput portion of the script can find out whether it is the restarted execution by examining the `isRestarted` attribute

of the context argument (`inputContext.isRestarted`). The script is being restarted if and only if `(context.isRestarted === true)`.

Note that the input for the next stage is computed from the return value of the `getInput` script. Next stage input is written after the `getInput` stage finishes. Therefore, even the restarted `getInput` script is expected to return the same data. The map/reduce framework helps to ensure that no data is written twice.

However, if the `getInput` script is changing some additional data (for example, creating NetSuite records), it should contain code to handle duplicated processing. The script needs idempotent operations to ensure that these records are not created twice, if this is undesired.

Termination of Shuffle Stage

The shuffle stage does not contain any custom code, so if the shuffle stage job is forcefully terminated, it is later restarted and all the work is completely redone. There is no impact other than that the stage takes longer to finish.

Termination of Parallel Stages

Map and reduce stages can execute jobs in parallel, so they are considered parallel stages. An application restart will affect parallel stages in the same way. The following example covers impact of restart during the map stage. Note that termination of a reduce stage will behave very similarly.

The purpose of a map stage is to execute a map function on each key/value pair supplied by the previous stage (`getInput`). Multiple jobs participate in the map stage. Map jobs will claim key/value pairs (or a specific number of key/value pairs) for which the map function was not executed yet. The job sets a flag for these key/value pairs so that no other job can execute the map function on them. Then, the job sequentially executes the map function on the key/value pairs it flagged. The map stage is finished when the map function is executed on all key/value pairs.

The number of jobs that can participate on the map stage is unlimited. Only the maximum concurrency is limited. Initially, the number of map jobs is equal to the selected concurrency in the corresponding map/reduce script deployment. However, to prevent a single map/reduce task from monopolizing all computational resources in the account, each map job can yield itself to allow other jobs to execute. The yield creates an additional map job and the number of yields is unlimited.

Note: This is a different type of yield compared to yield in a SuiteScript 1.0 scheduled script. In SuiteScript 1.0, the yield happens in the middle of a script execution. In a map job, the yield can happen only between two map function executions, and not in the middle of one.

If a map job is forcefully terminated, it is later restarted. First, the job executes the map function on all key/value pairs that it took and did not mark finished before termination. It is the only map job that can execute the map function on those pairs. They cannot be taken by other map job. After those key/value pairs are processed, the map job continues normally (takes other unfinished key/value pairs and executes the map function on them).

In some cases, the map function can be re-executed on multiple key/value pairs. The number of pairs that a map function can re-execute will depend on the buffer size selected on the deployment page. The buffer size determines the number of key/value pairs originally taken in a batch. The job marks the batch as finished only when the map function is executed on all of them. Therefore, if the map job is forcefully terminated in the middle of the batch, the entire batch will be processed from the beginning when the map job is restarted.

Note that the map/reduce framework deletes all key/value pairs written from a partially-executed batch, so that they are not written out twice. Therefore, the map function does not need to check

whether or not `mapContext.write(options)` for a particular key/value has already been executed. However, if the map function is changing some additional data, it must also be designed to use idempotent operations. For example, if a map function created NetSuite records, the script should perform additional checks to ensure that these records are not created twice, if this is undesired.

To check if a map function execution is a part of a restarted batch, the script must examine the `isRestarted` attribute in the context argument (`mapContext.isRestarted`). The map function is in the restarted batch if and only if `(context.isRestarted === true)`.

Be aware that a restarted value of `true` is only an indication that some part of the script might have already been executed. Even if `context.isRestarted === true`, a map function could run on a particular key/value for the first time. For example, the map job was forcefully terminated after the map job took the key/value pair for processing, but before it executed the map function on it. This is more likely to occur if a high buffer value is set on the map/reduce deployment. For more information about the buffer, see [Minimizing the Risk of Data Duplication](#).

Termination of Summarize Stage

If the summarize stage job is forcefully terminated, it is later restarted. The summarize portion of the script can find out whether it is the restarted execution by examining the `isRestarted` attribute of the summary argument (`summaryContext.isRestarted`).

The script is being restarted if and only if `(summary.isRestarted === true)`.

Adding Logic to Handle Map/Reduce Restarts

Occasionally, a script failure may occur due to an application server restart. This could be due to a NetSuite update, NetSuite maintenance, or an unexpected failure of the execution environment.

Restarts can terminate an application forcefully at any moment. Therefore, robust scripts must account for restarts and be able to recover from an unexpected interruption.

In a map/reduce script, each restarted piece of the script will automatically delete any internal map/reduce data that this piece created (for example, the key/value pairs that drive the execution of a entire mapping task). However, you must develop your own code to handle any parts of the script that modify additional data (for example, creation of NetSuite records like sales orders), which is *never* automatically deleted.

See the following topics to learn more about how restarts and map/reduce script execution:

- [Example 3: Design of a Robust Map/Reduce Script](#)
- [Example 4: A Problematic Map/Reduce Script](#)
- [Example 5: A Robust Map/Reduce Script](#)
- [Execution of Restarted Map/Reduce Stages](#)

 **Note:** A map or reduce function can also be restarted if interrupted by an uncaught error. For the script to restart in this situation, you must use the `retryCount` option. For additional details, see [System Response After an Uncaught Error](#).

Example 3: Design of a Robust Map/Reduce Script

The following script is designed to detect restarts at particular stages in processing, and to hold logic to run in the event of a restart.

Consider this example as a basic template, where the comment `// I might do something differently` denotes implementation of a special function for each stage, to ensure that the script can repeat itself with the same result. Or, to run a recovery task, such as removing duplicate records.

The script includes a check on the `isRestarted` property for each entry point function. If the value of `isRestarted` is `true`, the example script shows a placeholder for invoking a function. This is meant as a placeholder where implementation of logic for protection against restarts and data errors could be inserted.

For more information about an interrupted map/reduce stage, see [Execution of Restarted Map/Reduce Stages](#).

```
define([], function(){
  return {
    getInputData: function (context)
    {
      if (context.isRestarted)
      {
        // I might do something differently
      }
      .
      .
      .
      return inputForNextStage;
    },
    map: function (context)
    {
      if (context.isRestarted)
      {
        // I might do something differently
      }
      .
      .
      .
    },
    reduce: function (context)
    {
      if (context.isRestarted)
      {
        // I might do something differently
      }
      .
      .
      .
    },
    summarize: function (summary)
    {
      if (summary.isRestarted)
      {
        // I might do something differently
      }
      .
      .
      .
    }
  }
})
```

```
});
```

Example 4: A Problematic Map/Reduce Script

The purpose of this script is to perform a search and process the results. However, it is not adequately prepared for an unexpected restart. The script still needs logic to help prevent an unrecoverable state and prevent creation of erroneous or duplicate data during re-execution.

In Example 4, if the script is forcefully interrupted during the map stage, some sales orders might be updated twice when the map function is re-executed. See [Example 5: A Robust Map/Reduce Script](#) for an improved example.

Note that the other stages in this script do not require improvement for handling a restart. If the get input stage is re-executed, the map/reduce framework ensures that each result of the search is passed to the map stage only one time. In this script, the getInput stage does not change any additional data, so no special restart logic is needed to ensure correct updates of getInput data. Likewise, the reduce and summarize stages do not change any additional data. They process only internal map/reduce data.

```
/**
 * @NApiVersion 2.0
 * @NScriptType mapreducescript
 */
define(['N/search', 'N/record'],
function(search, record){
    return {
        getInputData: function (context)
        {
            var filter1 = search.createFilter({
                name: 'mainline',
                operator: search.Operator.IS,
                values: true
            });
            var column1 = search.createColumn({name: 'recordtype'});
            var srch = search.create({
                type: search.Type.SALES_ORDER,
                filters: [filter1],
                columns: [column1]
            });
            return srch;
        },
        map: function (context)
        {
            var soEntry = JSON.parse(context.value);
            var so = record.load({
                type: soEntry.values.recordtype,
                id: context.key
            });
            // UPDATE so FIELDS
            so.save();

            context.write({
                key: soEntry.values.recordtype,
                value: context.key
            });
        },
    },
```

```

    reduce: function (context)
    {

        context.write({
            key: context.key,
            value: context.values.length
        });

    },
    summarize: function (summary)
    {
        var totalRecordsUpdated = 0;
        summary.output.iterator().each(function (key, value)
        {
            log.audit({
                title: key + ' records updated',
                details: value
            });
            totalRecordsUpdated += parseInt(value);
            return true;
        });
        log.audit({
            title: 'Total records updated',
            details: totalRecordsUpdated
        });
    }
}
});

```

Example 5: A Robust Map/Reduce Script

Comparing Example 5 to Example 4, a filter was added to the search in the `getInput` stage. The purpose is to filter out already processed sales orders. The filter makes it possible to re-execute the whole map/reduce task repeatedly, because when the whole task is re-executed, the additional filter ensures that only unprocessed sales orders will be returned from the input stage and not all sales orders.

There are also substantial improvements to the map function. In Example 5, if the `((context.isRestarted === false))` condition is met, the script knows it is the first execution of the map function for the current key/value pair. It won't need to perform any additional checks and can go directly to the sales order record update.

During the sales order update, an operation sets the `custbody_processed_flag` flag. The script performs a check on this flag only as necessary. If `(context.isRestarted === true)`, then the script looks up the appropriate processed flag value, and executes the sales order update only if it wasn't already updated.

Although the script includes more checks and lookups than example 4, the processing demand is light. To perform the check, the script uses a lookup method that doesn't load the full record. If the processed flag value is `true`, then the record is not loaded again.

In the map function, note that the `context.write(...)` statement is not in the if-statement body. It is because when a map function for a particular key/value pair is restarted, all these writes done in the previous execution of the map function are deleted. So there is no need to check which writes have or haven't been done.

The reduce function is not changed from Example 4. This reduce stage handles only the map/reduce internal data, and so the map/reduce framework ensures that even when the reduce function is

restarted for a particular key/value pair, only the writes from its last execution for the key/value pair are passed to the next stage.

The summarize function also didn't require improvement. However, it is a good practice to log any restarts. For example, to account for when the "Total records updated" entry appears twice in the execution log for a single map/reduce task execution.

```
/**
 * @NApiVersion 2.0
 * @NScriptType mapreducescript
 */
define(['N/search', 'N/record'],
function(search, record){
    return {
        getInputData: function (context)
        {
            var filter1 = search.createFilter({
                name: 'mainline',
                operator: search.Operator.IS,
                values: true
            });
            var filter2 = search.createFilter({
                name: 'custbody_processed_flag',
                operator: search.Operator.IS,
                values: false
            });
            var column1 = search.createColumn({name: 'recordtype'});
            var srch = search.create({
                type: search.Type.SALES_ORDER,
                filters: [filter1, filter2],
                columns: [column1]
            });
            return srch;
        },
        map: function (context)
        {
            var soEntry = JSON.parse(context.value);
            var alreadyProcessed = false;
            if (context.isRestarted)
            {
                var lookupResult = search.lookupFields({
                    type: soEntry.values.recordtype,
                    id: context.key,
                    columns: ['custbody_processed_flag']
                });
                alreadyProcessed = lookupResult.custbody_processed_flag;
            }
            if (!alreadyProcessed)
            {
                var so = record.load({
                    type: soEntry.values.recordtype,
                    id: context.key
                });
                // UPDATE so FIELDS
                so.setValue({
                    fieldId: 'custbody_processed_flag',
```



```

        value: true
    });
    so.save();
}

context.write({
    key: soEntry.values.recordtype,
    value: context.key
});

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

},
summarize: function (summary)
{
    if (summary.isRestarted)
    {
        log.audit({details: 'Summary stage is being restarted!'});
    }
    var totalRecordsUpdated = 0;
    summary.output.iterator().each(function (key, value)
    {
        log.audit({
            title: key + ' records updated',
            details: value
        });
        totalRecordsUpdated += parseInt(value);
        return true;
    });
    log.audit({
        title: 'Total records updated',
        details: totalRecordsUpdated
    });
}
});

```

SuiteScript 2.0 Map/Reduce Script Best Practices

- Writing Efficient Map and Reduce Functions
- Passing Search Data to the getInputData Stage
- Minimizing the Risk of Data Duplication

Writing Efficient Map and Reduce Functions

As described in [Hard Limits on Function Invocations](#), NetSuite imposes governance limits on single invocations of map, reduce, getInputData, and summarize functions:

- **map:** 1,000 usage units (the same as mass update scripts)
- **reduce:** 5,000 usage units
- **getInputData:** 10,000 usage units
- **summarize:** 10,000 usage units

If you are concerned about potential issues with these limits, review your script to make sure that your map and reduce functions are relatively lightweight. Your map and reduce functions should not include a long or complex series of actions. For example, consider a situation in which your map or reduce function loads and saves multiple records all at once. This approach might cause an issue with the limits described above. If your `getInputData` function returns a list of record IDs, a better approach might be to use the map function to load each record, update fields on the record, and save it.

If you have a script that performs a significantly more complex series of operations within a single function (such as loading and saving multiple records, or transforming multiple records), consider using a different script type, such as a scheduled script.

Passing Search Data to the `getInputData` Stage

In the `getInputData` stage, your script must return an object that can be transformed into a list of key/value pairs. A common approach is to use a search. If you decide to use this technique, note that the recommended approach is to have your function return either a `search.Search` object or an object reference to a saved search. By contrast, if you execute a search within the `getInputData` function and return the results (for example, as an array), there is a greater risk that the search will time out.

Instead, Oracle recommends using one of the following approaches:

- Return a search object. That is, return an object created using `search.create(options)` or `search.load(options)`.
- Return a search object reference. That is, return an `inputContext.ObjectRef` object that references a saved search.

In both cases, the time limit available to the search is more generous than it would be for a search executed within the function.

The following snippet shows how to return a search object:

```
function getInputData()
{
    return search.create({
        type: record.Type.INVOICE,
        filters: [['status', search.Operator.IS, 'open']],
        columns: ['entity'],
        title: 'Open Invoice Search'
    });
}
```

And the following snippet shows how to return a search object reference:

```
...
function getInputData {
{
    // Reference a saved search with internal ID 1234.

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

```
}  
...
```

For information about additional ways to return data, see [getInputData\(inputContext\)](#).

Minimizing the Risk of Data Duplication

A map/reduce script can be interrupted at any time by an application server disruption. Afterward, the script is restarted.

Depending on how the script is configured, when a map or reduce job starts again, it may attempt to retry processing for the same key/value pairs it had flagged for processing when the interruption occurred. Similarly, if an uncaught error disrupts the job, the system may retry processing for the pair that was being processed when the error occurred.



Note: For an overview of the system's behavior following an interruption, see [System Response After a Map/Reduce Interruption](#).

When a job is restarted, there is an inherent risk of data duplication. To minimize this risk, use the following guidance.

Add Logic for Handling Restarts

Every map/reduce script should be written in such a way that each entry point function checks to see whether the function has been previously invoked. To do this, use the `context.isRestarted` property, which exists for every map/reduce entry point. If the function has been restarted, the script should provide any logic needed to avoid duplicate processing. For examples, see [Adding Logic to Handle Map/Reduce Restarts](#).

Leave Buffer Size Set to 1

When you deploy a script, the deployment record includes a field called Buffer Size. The default value of this field is 1. In general, Oracle recommends that you leave this value set to the default.

The purpose of the Buffer Size field is to control how many key/value pairs are flagged for processing at one time, and how frequently a map or reduce job saves data about its progress. Setting this field to a higher value may have a small performance advantage. However, the disadvantage is that, if the job is interrupted by an application server restart, there is a greater likelihood of one or more key/value pairs being processed twice. For that reason, Oracle recommends leaving this value set to 1, particularly if the script is processing records.

For more details on this field, see [Buffer Size](#).

SuiteScript 2.0 Map/Reduce Script Entry Points and API

This topic includes two parts:

- [Map/Reduce Script Entry Points](#) — A summary of the four entry points available to a map/reduce script.
- [Map/Reduce Script API](#) — A summary of properties available through the map/reduce entry points.

Map/Reduce Script Entry Points

The map/reduce script type includes four entry points. These entry points let you control both the script's behavior and the data flow within the map/reduce stages. For an overview of the stages, see [SuiteScript 2.0 Map/Reduce Script Stages](#).

Entry Point	Context Object	Required/Optional	Description
getInputData(inputContext)	inputContext	Required	Marks the beginning of the map/reduce script execution. Invokes the input stage. This function is invoked one time in the execution of the script.
map(mapContext)	mapContext	Optional, but if this entry point is skipped, the reduce(reduceContext) entry point is required.	Invokes the map stage. If this entry point is used, the map function is invoked one time for each key/value pair provided by the getInputData(inputContext) function.
reduce(reduceContext)	reduceContext	Optional, but if this entry point is skipped, the map(mapContext) entry point is required.	Invokes the reduce stage. If this entry point is used, the reduce function is invoked one time for each key and list of values provided. Data is provided either by the map stage or, if the map stage is not used, by the getInputData stage.
summarize(summaryContext)	summaryContext	Optional	Invokes the summarize stage. If the summarize entry point is used, the summarize function is invoked one time in the execution of the script.

Map/Reduce Script API

The following tables describe properties that are available through the map/reduce entry points.

inputContext Object Members

The following members are called on [inputContext](#).

Member Type	Name	Return Type / Value Type	Description
Property	inputContext.isRestarted	Boolean	Indicates whether the current invocation of the getInputData(inputContext) function represents a restart.
Object	inputContext.ObjectRef	object	And object that contains the input data.

The following members are called on the [inputContext.ObjectRef](#).

Member Type	Name	Return Type / Value Type	Description
Property	ObjectRef.id	string number	The internal ID or script ID of the object. For example, this value could be a saved search ID.

Member Type	Name	Return Type / Value Type	Description
	ObjectRef.type	string	The object's type.

mapContext Object Members

The following members are called on [mapContext](#).

Member Type	Name	Return Type / Value Type	Description
Property	mapContext.isRestarted	Boolean	Indicates whether the current invocation of the map(mapContext) function represents a restart. If the value of isRestarted is true, then the function was invoked previously, but unsuccessfully, for the current key/value pair.
	mapContext.executionNo	property	Indicates whether the current invocation of the map(mapContext) function represents the first or a subsequent attempt to process the current key/value pair.
	mapContext.errors	iterator	Holds serialized errors that were thrown during previous attempts to execute the map(mapContext) function on the current key/value pair.
	mapContext.key	string	The key to be processed during the current invocation of the map(mapContext) function.
	mapContext.value	string	The value to be processed during the current invocation of the map(mapContext) function.
Method	mapContext.write(options)	void	Writes the map(mapContext) output as key/value pairs. This data is passed to the reduce stage, if the reduce entry point is used, or to the summarize stage.

reduceContext Object Members

The following members are called on [reduceContext](#).

Member Type	Name	Return Type / Value Type	Description
Property	reduceContext.isRestarted	Boolean	Indicates whether the current invocation of the reduce(reduceContext) function represents a restart. If the value of isRestarted is true, then the function was invoked previously, but unsuccessfully, for the current key/value pair.
	reduceContext.executionNo	number	Indicates whether the current invocation of the reduce(reduceContext) function represents the first or a subsequent attempt to process the current key/value pair.
	reduceContext.errors	iterator	Holds serialized errors that were thrown during previous attempts to execute the reduce(reduceContext) function on the current key and its associated values.

Member Type	Name	Return Type / Value Type	Description
	<code>reduceContext.key</code>	string	The input key to process during the reduce stage.
	<code>reduceContext.values</code>	string[]	The key to be processed during the current invocation of the <code>reduce(reduceContext)</code> function.
Method	<code>reduceContext.write(options)</code>	void	Writes the <code>reduce(reduceContext)</code> function as key/value pairs. This data is passed to the summarize stage.

summaryContext Object Members

The following members are called on the `summaryContext`.

Member Type	Name	Value Type	Description
Property	<code>summaryContext.isRestarted</code>	read-only Boolean	Indicates whether the current invocation of the <code>summarize(summaryContext)</code> function represents a restart. If the value of <code>isRestarted</code> is true, then the function was invoked previously, but unsuccessfully.
	<code>summaryContext.concurrency</code>	number	The maximum concurrency number when running the map/reduce script.
	<code>summaryContext.dateCreated</code>	Date	The time and day when the script began running.
	<code>summaryContext.seconds</code>	number	The total number of seconds that elapsed during the processing of the script.
	<code>summaryContext.usage</code>	number	The total number of usage units consumed during the processing of the script.
	<code>summaryContext.yields</code>	number	The total number of yields that occurred during the processing of the script.
	<code>summaryContext.inputSummary</code>	object	Object that contains data about the input stage.
	<code>summaryContext.mapSummary</code>	object	Object that contains data about the map stage.
	<code>summaryContext.reduceSummary</code>	object	Object that contains data about the reduce stage.
	<code>summaryContext.output</code>	iterator	Iterator that contains the keys and values saved as the output of the reduce stage.

inputSummary Object members

The following members are called on `summaryContext.inputSummary`.

Member Type	Name	Value Type	Description
Property	<code>inputSummary.dateCreated</code>	Date	The time and day when the <code>getInputData(inputContext)</code> function began running.

Member Type	Name	Value Type	Description
	<code>inputSummary.error</code>	string	Holds serialized errors thrown from the <code>getInputData(inputContext)</code> function.
	<code>inputSummary.seconds</code>	number	The total number of seconds that elapsed during execution of the <code>getInputData(inputContext)</code> function. This tally does not include idle time.
	<code>inputSummary.usage</code>	number	The total number of usage units consumed by processing of the <code>getInputData(inputContext)</code> function.

mapSummary Members

The following members are called on `summaryContext.mapSummary`.

Member Type	Name	Value Type	Description
Property	<code>mapSummary.concurrency</code>	number	Maximum concurrency number when running <code>map(mapContext)</code> .
	<code>mapSummary.dateCreated</code>	Date	The time and day when the first invocation of <code>map(mapContext)</code> function began.
	<code>mapSummary.errors</code>	iterator	Holds serialized errors thrown during the map stage.
	<code>mapSummary.keys</code>	iterator	Holds the keys passed to the map stage by the <code>getInputData</code> stage.
	<code>mapSummary.seconds</code>	number	The total number of seconds that elapsed during the map stage.
	<code>mapSummary.usage</code>	number	The total number of usage units consumed during the map stage.
	<code>mapSummary.yields</code>	number	The total number of yields that occurred during the map stage.

reduceSummary Members

The following members are called on `summaryContext.reduceSummary`.

Member Type	Name	Value Type	Description
Property	<code>reduceSummary.concurrency</code>	number	Maximum concurrency number when running <code>reduce(reduceContext)</code> .
	<code>reduceSummary.dateCreated</code>	Date	The time and day when the first invocation of the <code>reduce(reduceContext)</code> function began.
	<code>reduceSummary.errors</code>	iterator	Holds serialized errors thrown during the reduce stage.
	<code>reduceSummary.keys</code>	iterator	Holds the keys passed to the reduce stage.
	<code>reduceSummary.seconds</code>	number	The total number of seconds that elapsed during the reduce stage.
	<code>reduceSummary.usage</code>	number	The total number of usage units consumed during the reduce stage.

Member Type	Name	Value Type	Description
	<code>reduceSummary.yields</code>	number	The total number of yields that occurred during the reduce stage.

getInputData(inputContext)

Description	<p>Marks the beginning of the script's execution. The purpose of the input stage is to generate the input data.</p> <p>Executes when the <code>getInputData</code> entry point is triggered. This entry point is required. For information about the context object provided to this entry point, see inputContext.</p> <div> <p>Note: When returning a <code>inputContext.ObjectRef</code>, the supported types are <code>search</code> and <code>file</code>.</p> <p>Note: When <code>getInputData()</code> returns a data structure with a non-string value, before the value is stored, it is converted to a JSON string with <code>JSON.stringify()</code>.</p> </div> <p>Also see Passing Search Data to the getInputData Stage.</p>
Returns	<p>Array Object <code>search.Search</code> <code>inputContext.ObjectRef</code> <code>file.File</code> Object</p> <p>Examples:</p> <ul style="list-style-type: none"> Array <pre>[{a : 'b'}, {c : 'd'}]</pre> Object <pre>{ a: {...}, b: {...} }</pre> <code>search.Search</code> Object <pre>search.load({ id: 1234 })</pre> Object Reference <pre>{ type: 'search', id: 1234 }</pre> A <code>file.File</code> Object <pre>file.load({ id: 1234 })</pre> Object Reference <pre>{ type: 'file', path: '/SuiteScripts/data/names.txt' }</pre>

	<div> } </div> <div>  Note: When using an Object Reference to a file, you must use an absolute path or bundle virtual path. Relative paths are not permitted. </div>
Since	Version 2015 Release 2

Errors

When an error is thrown in this function, the job proceeds to the [summarize\(summaryContext\)](#) function. The serialized error is encapsulated in the [inputSummary.error](#) property.

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code
...
function getInputData {
{

    // Reference a saved search that returns a list of NetSuite records that
    // require processing - for example, sales orders that are pending fulfillment.

    return {
        type: 'search',
        id: 1234
    };
}
...
// Add additional code
```

inputContext

Object Description	This object passed to the getInputData(inputContext) entry point function.
Since	Version 2016 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code
...
function getInputData {
{

    // Reference a saved search that returns a list of NetSuite records that
    // require processing - for example, sales orders that are pending fulfillment.

    return {
        type: 'search',
```

```

        id: 1234
    };
}
...
// Add additional code

```

inputContext.isRestarted

Property Description	Indicates whether the current invocation of the getInputData(inputContext) function is the first. Typically, the getInput function is invoked one time only. However, if the function is interrupted by an application server restart, the system restarts the getInputData function. If the value of this property is <code>true</code> , the getInputData(inputContext) has been restarted. You can use this property to help make your script more robust.
Type	read-only Boolean
Since	Version 2016 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
function getInputData(context)
{
    if (context.isRestarted)
    {
        log.debug('GET_INPUT isRestarted', 'YES');
    }
    else
    {
        log.debug('GET_INPUT isRestarted', 'NO');
    }

    var extractSearch = search.load({ id: 'customsearch35' });
    return extractSearch;
}
...

```

inputContext.ObjectRef

Object Description	References the object that contains the input data. For example, a reference to a saved search. You can use getInputData(inputContext) to return this object.
	<div>  Note: The only supported object types are <code>search</code> and <code>file</code>. </div>
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...

```

```

{
  type: 'search',
  id: 1234 //search internal id
}
...

```

ObjectRef.id

Property Description	The internal ID or script ID of the object. For example, the saved search ID.
Type	string number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
{
  type: 'search',
  id: 1234 //search internal id
}
...

```

ObjectRef.type

Property Description	The object's type.
Type	string
Values	'search'
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
{
  type: 'search',
  id: 1234 //search internal id
}
...

```

map(mapContext)

Description	<p>Executes when the <code>map</code> entry point is triggered.</p> <p>The logic in your map function is applied to each key/value pair that is provided by the <code>getInputData</code> stage. One key/value pair is processed per function invocation, then the function is invoked again for the next pair.</p> <p>The output of the map stage is another set of key/value pairs. During the shuffle stage that always follows, these pairs are automatically grouped by key.</p>
-------------	---

	For information about the context object provided to this entry point, see mapContext .
Returns	Void
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description
mapContext	Object	Required	Object that contains: <ul style="list-style-type: none"> ■ The key/value pairs to process during the map stage. ■ Logic that lets you save data to pass to the reduce stage. ■ Other properties you can use within the map function.

Errors

When an error is thrown, the behavior of the job varies depending on the setting of the [retryCount](#) configuration option.

If the function has been restarted for a key/value pair that it previously attempted to process, any errors logged during prior attempts can be accessed through [mapContext.errors](#).

In the summary stage, you can review all map stage errors by using the [reduceSummary.errors](#).

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code
...
function map(context)
{
  for (var i = 0; context.value && i < context.value.length; i++)
    if (context.value[i] !== ' ' && !PUNCTUATION_REGEXP.test(context.value[i]))
    {
      context.write({
        key: context.value[i],
        value: 1
      });
    }
}
...
// Add additional code
```

mapContext

Object Description	Object that contains: <ul style="list-style-type: none"> ■ The key/value pairs to process during the map stage. ■ Logic that lets you save data to pass to the reduce stage.
---------------------------	--

	<ul style="list-style-type: none"> Other properties you can use within the map function. For a complete list of this object's methods and properties, see mapContext Object Members .
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
function map(context)
{
    for (var i = 0; context.value && i < context.value.length; i++)
        if (context.value[i] !== ' ' && !PUNCTUATION_REGEXP.test(context.value[i]))
        {
            context.write({
                key: context.value[i],
                value: 1
            });
        }
}
...
```

mapContext.isRestarted

Property Description	<p>Indicates whether the map(mapContext) function has been invoked previously for the current key/value pair.</p> <p>For an overview of events that can cause a restart, see System Response After a Map/Reduce Interruption.</p> <p>When the map function is restarted for a key/value pair, data previously written by the incomplete function is deleted. However, some of the function's logic might have been executed before the function invocation was interrupted. For that reason, if the <code>mapContext.isRestarted</code> value is true, your script should take the necessary actions to avoid duplicate processing. For examples, see Adding Logic to Handle Map/Reduce Restarts.</p> <p>Related properties include mapContext.executionNo and mapContext.errors.</p>
Type	Boolean
Since	Version 2016 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code.
...
function map(context) {
    if (context.isRestarted)
    {
        // Add logic designed to assess how much processing was completed for this key/value pair and react accordingly.
    }
    else
    {
        // Let full processing continue for the key/value pair.
    }
}
```

```

    }
    ...
    // Add additional code.

```

mapContext.executionNo

Property Description	<p>Indicates whether the current invocation of the map(mapContext) function is the first or a subsequent invocation for the current key/value pair.</p> <p>For an overview of events that can cause a restart, see System Response After a Map/Reduce Interruption.</p> <p>When the map function is restarted for a key/value pair, data previously written by the incomplete function is deleted. However, some of the function's logic might have been executed before the function invocation was interrupted. For that reason, you may want to use the <code>mapContext.executionNo</code> property to provide logic designed to avoid duplicate processing. For examples of how to write a robust map/reduce script, see Adding Logic to Handle Map/Reduce Restarts.</p> <p>Related properties include mapContext.isRestarted and mapContext.errors.</p>
Type	number
Since	Version 2018 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

// Add additional code
...

if (context.executionNo === 1){
    // Permit full processing of the key/value pair.
}

else if (context.executionNo === 2){
    // Take steps to check whether any processing was previously completed.
}

else {
    // Take other steps that might be necessary in the case of more than
    // one previous attempt.
}
...
// Add additional code

```

mapContext.errors

Property Description	<p>Holds serialized errors that were thrown during previous attempts to execute the map function on the current key/value pair.</p> <p>This iterator may also hold the <code>SSS_APP_SERVER_RESTART</code> error code, which is recorded if the function is interrupted by an application server restart.</p> <p>For an overview of events that can cause the map function to be invoked more than once for a key/value pair, see System Response After a Map/Reduce Interruption.</p>
Type	iterator
Since	Version 2018 Release 1

Members

Member	Type	Required/Optional	Description
iterator().each(parameters)	function	required	Executes one time for each error.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, error, executionNo) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents the key/value pair that the map function was attempting to process when the error occurred.
error	string	optional	A serialization of the error thrown.
executionNo	number	optional	Indicates whether the error occurred during the first or a subsequent attempt to process the key/value pair.

Syntax

The following snippets shows three ways you could use this iterator.

This code is not a functional example. For a complete script sample, see [Map/Reduce Script Samples](#).

```
//Add additional code
...

// Create a log entry showing each full serialized error, and the corresponding key.

context.errors.iterator().each(function (key, error, executionNo){
    log.error({
        title: 'Map error for key: ' + key + ', execution no. ' + executionNo,
        details: error
    });
    return true;
});

// Log only the name and description of each error thrown.

context.errors.iterator().each(function (key, error, executionNo){
    var errorObject = JSON.parse(error);
    log.error({
        title: 'Reduce error for key ' + key + ', execution no. ' + executionNo,
        details: errorObject.name + ': ' + errorObject.message
    });
    return true;
});
```

```
// Calculate and log the number of errors encountered.

var errorCount = 0;
context.errors.iterator().each(function() {
    errorCount ++;
    return true;
});

log.audit ({
    title: 'Errors for map key: ' + context.key,
    details: 'Total number of errors: ' + errorCount
});
```

mapContext.key

Property Description	<p>The key to be processed during the map stage.</p> <ul style="list-style-type: none"> ■ If the input type is an array, the key is the index of the element. ■ If the input type is an object, the key is the key in the object. ■ If the input type is a result set, the key is the internal ID of the result. If the search result has no internal ID, the key is the index of the search result. <div>  Note: Each key cannot exceed 4000 bytes. </div>
Type	string
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code
...

context.write({
    key: context.value[i],
    value: 1
});
...
// Add additional code
```

mapContext.value

Property Description	<p>The value to be processed during the map stage.</p> <ul style="list-style-type: none"> ■ If the input type is an array, the mapContext.value is the value in the element. ■ If the input type is an object, the mapContext.value is the value in the object. ■ If the input type is a result set, the the mapContext.value is a <code>search.Result</code> object converted to a JSON string by using <code>JSON.stringify()</code>.
-----------------------------	--

	 Note: Each value cannot exceed 1 megabyte.
Type	string
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
// Add additional code.
...

// Assume that the search result is a list of phone call records. This snippet parses the results
// and uses the values of the title and the message fields from each record.

var searchResult = JSON.parse(context.value);

var title = searchResult.values.title;
var message = searchResult.values.message;

...
// Add additional code.
```

mapContext.write(options)

Method Description	Writes the key/value pairs to be passed to the shuffle and then the reduce stage. If your script includes both a map and reduce function, you must use this method in order for the reduce function to be invoked.
Returns	Void
Since	Version 2015 Release 2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description
options.key	String or object. However, note that if you provide an object, the system calls JSON.stringify() on your input.	required	The key to write
options.value		required	The value to write

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
function map(context)
{
    for (var i = 0; context.value && i < context.value.length; i++)
        if (context.value[i] !== ' ' && !PUNCTUATION_REGEXP.test(context.value[i]))
        {
```

```

        context.write({
            key: context.value[i],
            value: 1
        });
    }
}
...

```

reduce(reduceContext)

Description	<p>Executes when the reduce entry point is triggered. The logic in your reduce function is applied to each key, and its corresponding list of value. Only one key, with its corresponding values, is processed per function invocation. The function is invoked again for the next key and corresponding set of values. Data is provided to the reduce stage by one of the following:</p> <ul style="list-style-type: none"> ■ The getInputData stage — if your script has no map function. ■ The shuffle stage — if your script uses a map function. The shuffle stage follows the map stage. Its purpose is to sort data from the map stage by key. <p>For information about the context object provided to this entry point, see reduceContext.</p>
Returns	Void
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description
reduceContext	Object	Required	<p>Object that contains:</p> <ul style="list-style-type: none"> ■ The data to process during the reduce stage. ■ Logic that lets you save data to pass to the summarize stage. ■ Other properties you can use within the reduce function.

Errors

When an error is thrown, the behavior of the job varies depending on the setting of the [retryCount](#) configuration option.

If the function has been restarted for a key/value pair that it previously attempted to process, any errors logged during prior attempts can be accessed through [mapContext.errors](#).

In the summary stage, you can review all map stage errors by using the [reduceSummary.errors](#).

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
function reduce(context)
{
    context.write({
        key: context.key ,

```

```

        value: context.values.length
    });
}
...

```

reduceContext

Object Description	Contains the key/value pairs to process during the reduce stage.
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

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

```

reduceContext.isRestarted

Property Description	<p>Indicates whether the <code>reduce(reduceContext)</code> function has been invoked previously for the current key and values.</p> <p>For an overview of events that can cause a restart, see System Response After a Map/Reduce Interruption.</p> <p>When the reduce function is restarted for a key/value pair, data previously written by the incomplete function is deleted. However, some of the function's logic might have been executed before the function invocation was interrupted. For that reason, if the <code>reduceContext.isRestarted</code> property is true, your script should take the necessary actions to avoid duplicate processing. For examples, see Adding Logic to Handle Map/Reduce Restarts.</p> <p>Related properties include <code>reduceContext.executionNo</code> and <code>reduceContext.errors</code>.</p>
Type	Boolean
Since	Version 2016 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

// Add additional code.
...
function reduce (context) {
    if (context.isRestarted)
    {
        // Add logic designed to assess how much processing was completed for this key
        // and react accordingly.
    }
}

```

```

    else
    {
        // Let full processing continue for the current key and its values.
    }
    ...
    // Add additional code.

```

reduceContext.executionNo

Property Description	<p>Indicates whether the current invocation of the <code>reduce(reduceContext)</code> function is the first, second, third, or fourth for the current key and its values.</p> <p>For an overview of events that can cause a restart, see System Response After a Map/Reduce Interruption.</p> <p>When the reduce function is restarted for a key, data previously written by the incomplete function is deleted. However, some of the function's logic might have been executed before the function invocation was interrupted. For that reason, you may want to use the <code>reduceContext.executionNo</code> property to provide logic designed to avoid duplicate processing. For examples of how to write a robust map/reduce script, see Adding Logic to Handle Map/Reduce Restarts.</p> <p>Related properties include <code>reduceContext.isRestarted</code> and <code>reduceContext.errors</code>.</p>
Type	number
Since	Version 2018 Release 1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

// Add additional code
...

if (context.executionNo === 1){
    // Permit full processing of the key/value pair.
}

else if (context.executionNo === 2){
    // Take steps to check whether any processing was previously completed.
}

else {
    // Take other steps that might be necessary in the case of more than
    // one previous attempt.
}
...
// Add additional code

```

reduceContext.errors

Property Description	<p>Holds serialized errors that were thrown during previous attempts to execute the reduce function on the current key and its values.</p> <p>This iterator may also hold the <code>SSS_APP_SERVER_RESTART</code> error code, which is recorded if the function is interrupted by an application server restart.</p> <p>For an overview of events that can cause the map function to be invoked more than once for a key/value pair, see System Response After a Map/Reduce Interruption.</p>
-----------------------------	---

Type	iterator
Since	Version 2018 Release 1

Members

Member	Type	Required/Optional	Description
iterator().each(Parameters)	function	required	Executes one time for each error.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, error, executionNo) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents the key/value pair that the reduce function was attempting to process when the error occurred.
error	string	optional	A serialization of the error thrown.
executionNo	number	optional	Indicates whether the error occurred during the first or a subsequent attempt to process the key.

Syntax

The following snippets shows three ways you could use this iterator.

This code is not a functional example. For a complete script sample, see [Map/Reduce Script Samples](#).

```
//Add additional code
...

// Create a log entry showing each full serialized error, and the corresponding key.
context.errors.iterator().each(function (key, error, executionNo){
    log.error({
        title: 'Reduce error for key: ' + key + ', execution no ' + executionNo,
        details: error
    });
    return true;
});

// Log only the name and description of each error thrown.
context.errors.iterator().each(function (key, error, executionNo){
    var errorObject = JSON.parse(error);
    log.error({
        title: 'Reduce error for key ' + key + ', execution no. ' + executionNo,
        details: errorObject.name + ': ' + errorObject.message
    });
});
```

```

    });
    return true;
  });

  // Calculate and log the number of errors encountered.

  var errorCount = 0;
  context.errors.iterator().each(function() {
    errorCount++;
    return true;
  });

  log.audit ({
    title: 'Errors for reduce key: ' + context.key,
    details: 'Total number of errors: ' + errorCount
  });

```

reduceContext.key

Property Description	<p>When the map/reduce process includes a map stage, the reduce keys are the keys written by mapContext.write(options). When the map stage is skipped, the reduce keys are provided by the getInputData stage:</p> <ul style="list-style-type: none"> ■ If the input type is an array, the key is the index of the element. ■ If the input type is an object, the key is the key in the object. ■ If the input type is a result set, the key is the internal ID of the result. <div>  Note: Each key cannot exceed 4000 bytes. </div>
Type	string
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

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

```

reduceContext.values

Property Description	<p>This array holds all values associated with a unique key that was passed to the reduce stage for processing. These values are listed in lexicographical order. When the map/reduce process includes a map stage, the key/value pairs passed to the reduce stage are derived from the values written by mapContext.write(options).</p>
-----------------------------	--

	<p>When the map stage is skipped, the values are determined by the getInputData stage:</p> <ul style="list-style-type: none"> ■ If the input type is an array, it is the value in the element. ■ If the input type is an object, it is the value in the object. ■ If the input type is a result set, the value is a <code>search.Result</code> object converted to a JSON string with <code>JSON.stringify()</code>. <div>  Note: Each value cannot exceed 1 megabyte. </div>
Type	string[]
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

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

reduceContext.write(options)

Method Description	Writes key/value pairs.
Returns	Void
Since	Version 2015 Release 2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.key	String or object. However, note that if you provide an object, the system calls <code>JSON.stringify()</code> on your input.	required	The key to write.
options.value		required	The value to write

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
function reduce(context)
{
    context.write({
        key: context.key,
```

```

        value: context.values.length
    });
}
...

```

summarize(summaryContext)

Description	Executes when the <code>summarize</code> entry point is triggered. When you add custom logic to this entry point function, that logic is applied to the result set. For information about the context object provided to this entry point, see summaryContext .
Returns	Void
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description
summary	summaryContext	Required	Holds statistics regarding the execution of a map/reduce script.

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
function summarize(summary)
{
    var type = summary.toString();
    log.audit(type + ' Usage Consumed', summary.usage);
    log.audit(type + ' Concurrency Number ', summary.concurrency);
    log.audit(type + ' Number of Yields', summary.yields);
    var contents = '';
    summary.output.iterator().each(function (key, value)
    {
        contents += (key + ' ' + value + '\n');
        return true;
    });
}
...

```

summaryContext

Object Description	Holds statistics regarding execution of a map/reduce script.
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...

```



```
function summarize(summary)
{
    var type = summary.toString();
    log.audit(type + ' Usage Consumed', summary.usage);
    log.audit(type + ' Concurrency Number ', summary.concurrency);
    log.audit(type + ' Number of Yields', summary.yields);
    var contents = '';
    summary.output.iterator().each(function (key, value)
    {
        contents += (key + ' ' + value + '\n');
        return true;
    });
    ...
}
```

summaryContext.isRestarted

Property Description	Indicates whether the <code>summarize(summaryContext)</code> function was invoked again. To reduce negative impact to map/reduce processing if the Java virtual machine (JVM) restarts, NetSuite automatically restarts the current summary function. Summary data previously written by the incomplete function is deleted. If the value is <code>true</code> , the current process invoked by <code>summarize(summaryContext)</code> was restarted. You can use this information to help you write a more robust map/reduce script that is designed to continue interrupted work as necessary.
Type	read-only Boolean
Since	Version 2016 Release 1

Syntax

```
...
function summarize(summary) {
    if (summary.isRestarted)
    {
        log.debug('SUMMARY isRestarted', 'YES');
    }
    else
    {
        log.debug('SUMMARY isRestarted', 'NO');
    }
    log.debug('summarize', JSON.stringify(summary));
    ...
}
```

summaryContext.concurrency

Property Description	The maximum concurrency number when executing parallel tasks for the map/reduce script. <div>  Note: This number may be less than the number allocated on the script deployment. For example, tasks that remained in the pending state are not reflected in this number. </div>
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(type + ' Concurrency Number ', summary.concurrency);
...
```

summaryContext.dateCreated

Property Description	The time and day when the map/reduce script began running.
Type	Date
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(type + ' Creation Date', summary.dateCreated);
...
```

summaryContext.seconds

Property Description	Total seconds elapsed when running the map/reduce script.
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(type + ' Total seconds elapsed', summary.seconds);
...
```

summaryContext.usage

Property Description	Total number of usage units consumed when running the map/reduce script.
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(type + ' Usage Consumed', summary.usage);
...
```

summaryContext.yields

Property Description	Total number of yields when running the map/reduce script.
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(type + ' Number of Yields', summary.yields);
...
```

summaryContext.inputSummary

Object Description	Holds statistics regarding the input stage.
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
logMetrics(summary.inputSummary);
log.error('Input Error', summary.inputSummary.error);
...
```

inputSummary.dateCreated

Property Description	The time and day when <code>getInputData(inputContext)</code> began running.
Type	Date
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Creation Date', summary.inputSummary.dateCreated);
...
```

inputSummary.seconds

Property Description	Total seconds elapsed when running <code>getInputData(inputContext)</code> (does not include idle time).
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Time Elapsed', summary.inputSummary.seconds);
...
```

inputSummary.usage

Property Description	Total number of usage units consumed when running <code>getInputData(inputContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Usage', summary.inputSummary.usage);
...
```

inputSummary.error

Property Description	If applicable, holds a serialized error that is thrown from <code>getInputData(inputContext)</code> .
Type	string
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
logMetrics(summary.inputSummary);
log.error('Input Error', summary.inputSummary.error);
...
```

summaryContext.mapSummary

Object Description	Holds statistics regarding the map stage
---------------------------	--

Since	Version 2015 Release 2
-------	------------------------

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
logMetrics(summary.mapSummary);
var mapKeys = [];
summary.mapSummary.keys.iterator().each(function (key)
{
    mapKeys.push(key);
    return true;
});
log.audit('MAP keys processed', mapKeys);
summary.mapSummary.errors.iterator().each(function (key, error)
{
    log.error('Map Error for key: ' + key, error);
    return true;
});
...
```

mapSummary.concurrency

Property Description	The maximum concurrency number for executing parallel tasks during the map stage.  Note: This number may be less than the concurrency number allocated on the script deployment. For example, tasks that remained in the pending state are not reflected in this number.
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit('Concurrency', summary.mapSummary.concurrency);
...
```

mapSummary.dateCreated

Property Description	The time and day when map(mapContext) began running.
Type	Date
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Creation Date', summary.mapSummary.dateCreated);
...
```

mapSummary.keys

Property Description	Holds the keys that were passed to the map stage by the getInputData stage. Note that keys are sorted in the following way: <ul style="list-style-type: none"> ■ If all values are numeric, values are listed in numeric order. ■ If one or more values are strings, values are listed in lexicographical order.
Type	iterator
Since	Version 2015 Release 2

Members

Member	Type	Required/Optional	Description
iterator().each(parameters)	function	required	Executes one time for each key.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, executionCount, completionState) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents a key that was passed to the map stage.
executionCount	number	optional	The number of times the map function was invoked for the key.
completionState	string	optional	A setting that indicates whether the map function was executed successfully for the key. Possible values are 'COMPLETE', 'FAILED', and 'PENDING.' The system uses 'PENDING' if the script exited the stage before trying to process the key. This behavior can occur when <code>exitOnError</code> is set to true.

Syntax

The following snippets show three ways you could use this iterator.

This code is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
//Add additional code.
...

// Log all keys from the map stage.
```

```

var mapKeys = [];
summary.mapSummary.keys.iterator().each(function (key){
    mapKeys.push(key);
    return true;
});

log.debug({
    title: 'Map stage keys',
    details: mapKeys
});

// Create a log entry showing whether the map function was executed succesfully for each key.

summary.mapSummary.keys.iterator().each(function (key, executionCount, completionState){

log.debug({
    title: 'Map key ' + key,
    details: 'Outcome for map key ' + key + ': ' + completionState + ' // Number of attempts used: ' + executionCount
});

return true;

});

// Create a log entry showing the total number of keys for which the map function executed successfully.

var mapKeysProcessed = 0;
summary.mapSummary.keys.iterator().each(function (key, executionCount, completionState){

if (completionState === 'COMPLETE'){
    mapKeysProcessed++;
}

return true;

});

log.debug({
    title: 'Map key statistics',
    details: 'Total number of map keys processed successfully: ' + mapKeysProcessed
});

```

mapSummary.seconds

Property Description	Total seconds elapsed when running <code>map(mapContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Time Elapsed', summary.mapSummary.seconds);
...
```

mapSummary.usage

Property Description	Total number of usage units consumed when running <code>map(mapContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Usage', summary.mapSummary.usage);
...
```

mapSummary.yields

Property Description	Total number of times yields when running <code>map(mapContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Total Yields', summary.mapSummary.yields);
...
```

mapSummary.errors

Property Description	Holds all serialized errors thrown from the <code>map(mapContext)</code> function. May also hold the <code>SSS_APP_SERVER_RESTART</code> error code, which is recorded in the event of an application server restart.
Type	iterator
Since	Version 2015 Release 2

Members

Member	Type	Required/Optional	Description
<code>iterator().each(parameters)</code>	function	required	Executes one time for each error.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, error, executionNo) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents a key that was passed to the map stage for processing.
error	string	optional	The error thrown during processing of the corresponding key.
executionNo	number	optional	Indicates whether the error occurred during the first, second, third, or fourth attempt to process the key.

Syntax

The following snippets show three ways you could use this iterator.

This code is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
//Add additional code
...

// Create a log entry showing each full serialized error.

summary.mapSummary.errors.iterator().each(function (key, error, executionNo){
    log.error({
        title: 'Map error for key: ' + key + ", execution no. " + executionNo,
        details: error
    });
    return true;
});

// Log only the name and description of each error thrown.

summary.mapSummary.errors.iterator().each(function (key, error, executionNo){
    var errorObject = JSON.parse(error);
    log.error({
        title: Map error for key: ' + key + ", execution no. ' + executionNo,
        details: errorObject.name + ': ' + errorObject.message
    });
    return true;
});

// Calculate and log the total number of errors encountered during the map stage.

var errorCount = 0;
summary.mapSummary.errors.iterator().each(function() {
    errorCount ++;
```

```

        return true;
    });

    log.audit ({
        title: 'Map stage errors',
        details: 'Total number of errors: ' + errorCount
    });

    ...
    //Add additional code

```

summaryContext.reduceSummary

Object Description	Holds statistics regarding the reduce stage.
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
summary.reduceSummary.errors.iterator().each(function (key, error)
{
    log.error('Reduce Error for key: ' + key, error);
    return true;
});
...

```

reduceSummary.concurrency

Property Description	The maximum concurrency number for executing parallel tasks during the reduce stage. <div>  Note: This number may be less than the concurrency number allocated on the script deployment. For example, tasks that remained in the pending state are not reflected in this number. </div>
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```

...
log.audit('Concurrency', summary.reduceSummary.concurrency);
...

```

reduceSummary.dateCreated

Property Description	The time and day when <code>reduce(reduceContext)</code> began running.
-----------------------------	---

Type	Date
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Creation Date', summary.reduceSummary.dateCreated);
...
```

reduceSummary.keys

Property Description	Holds the keys that were passed to the reduce stage, either by the map stage, if it was used, or by the getInputData stage. These keys are listed in lexicographical order .
Type	iterator
Since	Version 2015 Release 2

Members

Member	Type	Required/Optional	Description
iterator().each(parameters)	function	required	Executes one time for each key.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, executionCount, completionState) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents a key that was passed to the reduce stage.
executionCount	number	optional	The number of times the reduce function was invoked for the key.
completionState	string	optional	A setting that indicates whether the map function was executed successfully for the key. Possible values are 'COMPLETE', 'FAILED', and 'PENDING.' The system uses 'PENDING' if the script exited the stage before trying to process the key. This behavior can occur when <code>exitOnError</code> is set to true.

Syntax

The following snippets show a variety of ways you could use the `reduceSummary.keys` iterator. The code in the section does not comprise not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
//Add additional code.
```

```

...

// Create a single log entry listing all of the keys that were passed to the reduce stage.

var reduceKeys = [];

summary.reduceSummary.keys.iterator().each(function (key){
    reduceKeys.push(key);
    return true;
});

log.debug({
    title: 'Reduce stage keys',
    details: reduceKeys
});

// Create a log entry for each key. The entry shows whether the reduce function was executed successfully for that key.

summary.reduceSummary.keys.iterator().each(function (key, executionCount, completionState){

    log.debug({
        title: 'Reduce key ' + key,
        details: 'Outcome for reduce key ' + key + ': ' + completionState + ' // Number of attempts used: ' + executionCount
    });

    return true;

});

// creates a single log entry showing the total number of keys for which the reduce function was invoked successfully.

var reduceKeysProcessed = 0;
summary.reduceSummary.keys.iterator().each(function (key, executionCount, completionState){

    if (completionState === 'COMPLETE'){
        reduceKeysProcessed++;
    }

    return true;

});

log.debug({
    title: 'Reduce key statistics',
    details: 'Total number of reduce keys processed successfully: ' + reduceKeysProcessed
});

...
//Add additional code.

```

reduceSummary.seconds

Property Description	Total seconds elapsed when running <code>reduce(reduceContext)</code> .
----------------------	---

Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Time Elapsed', summary.reduceSummary.seconds);
...
```

reduceSummary.usage

Property Description	Total number of usage units consumed when running <code>reduce(reduceContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Usage', summary.mapSummary.usage);
...
```

reduceSummary.yields

Property Description	Total number of times yields when running <code>reduce(reduceContext)</code> .
Type	number
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Map/Reduce Script Samples](#).

```
...
log.audit(' Total Yields', summary.reduceSummary.yields);
...
```

reduceSummary.errors

Property Description	Holds all serialized errors thrown from the <code>reduce(reduceContext)</code> function. May also hold the <code>SSS_APP_SERVER_RESTART</code> error code, which is recorded in the event of an application server restart.
Type	iterator
Since	Version 2015 Release 2

Members

Member	Type	Required/Optional	Description
iterator().each(parameters)	function	required	Executes one time for each error.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, error, executionNo) See also functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	Represents a key that was passed to the reduce stage for processing.
error	string	optional	A serialization of the error thrown.
executionNo	number	optional	Indicates whether the error occurred during the first, second, third, or fourth attempt to process the key.

Syntax

The following snippets shows three ways you could use this iterator.

This code is not a functional example. For a complete script sample, see [Map/Reduce Script Samples](#).

```
//Add additional code
...

// Create a log entry showing each full serialized error.

summary.reduceSummary.errors.iterator().each(function (key, error, executionNo){
  log.error({
    title: 'Reduce error for key: ' + key + ', execution no. ' + executionNo,
    details: error
  });
  return true;
});

// Log only the name and description of each error thrown.

summary.reduceSummary.errors.iterator().each(function (key, error){
  var errorObject = JSON.parse(error);
  log.error({
    title: 'Reduce error for key: ' + key + ', execution no. ' + executionNo,
    details: errorObject.name + ': ' + errorObject.message
  });
  return true;
});
```

```
// Calculate and log the total number of errors encountered during the reduce stage.

var errorCount = 0;
summary.reduceSummary.errors.iterator().each(function() {
    errorCount ++;
    return true;
});

log.audit ({
    title: 'Reduce stage errors',
    details: 'Total number of errors: ' + errorCount
});
```

summaryContext.output

Property Description	Iterator that provides keys and values that are saved as output during the reduce stage. Note that keys are listed in lexicographical order .
Type	iterator
Since	Version 2015 Release 2

Members

Member	Type	Required/Optional	Description
iterator().each(parameters)	function	required	Executes one time for each key/value pair.

parameters

Member	Type	Required/Optional	Description
iteratorFunction(key, value). See functionParameters .	function	required	Provides logic to be executed during each iteration.

functionParameters

Parameter	Type	Required/Optional	Description
key	string	optional	The key saved at the end of the reduce stage by using the reduceContext.write(options) method.
value	string	optional	The value saved at the end of the reduce stage by using the reduceContext.write(options) method.

Syntax

```
// Add additional code
...

// Create a variable to track how many key/value pairs were written.
var totalRecordsUpdated = 0;

// If the number of key/value pairs is expected to be manageable, log
// each one.
```

```
summary.output.iterator().each(function (key, value){
    log.audit({
        title: ' summary.output.iterator',
        details: 'key: ' + key + ' / value: ' + value
    });
    totalRecordsUpdated++;
    return true;
});

// Create a log entry showing the number of

log.debug({
    title: 'Total records updated',
    details: totalRecordsUpdated
});
...
// Add additional code.
```

SuiteScript 2.0 Mass Update Script Type

Mass update scripts allow you to programmatically perform custom updates to fields that are not available through general mass updates. Mass update scripts can run complex calculations, as defined in your script, across records. Mass update scripts are started on demand, and you cannot prioritize them or schedule them to run at specific times. If you want your script to run at a specific time, consider using a scheduled script or map/reduce script instead of a mass update script. For more information, see [SuiteScript 2.0 Scheduled Script Type](#) and [SuiteScript 2.0 Map/Reduce Script Type](#).

Mass Update scripts are executed on the server when you click the Perform Update button on the Mass Update Preview Results page.

Be aware that updates made to records during a custom mass update can trigger user event scripts if there are user event scripts associated with the records being updated.

For information about scripting with mass update scripts, see [SuiteScript 2.0 Mass Update Script Entry Points](#).

Mass Update Script Sample

The following code updates the probability field of all existing records to 61%.

 **Note:** From the script deployment record, ensure that the **Applies To** field is set to **Opportunity**.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.0
 * @NScriptType MassUpdateScript
 */
define(['N/record'],
    function(record) {
        function each(params) {
```



```

    // Set the probability to 61%
    var recOpportunity = record.load({
        type: params.type,
        id: params.id
    });
    recOpportunity.setValue('probability', 61);
    recOpportunity.save();
}
return {
    each: each
};
});

```

SuiteScript 2.0 Mass Update Script Entry Points

Script Entry Point	
each	Iterates through all applicable records so that you can apply logic to each record. See each for parameters and syntax.

each

Description	Iterates through each applicable record.
Returns	void
Since	Version 2016 Release 1

Parameters

Note: The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.id</code>	number	The ID of the record being processed by the mass update.	Version 2016 Release 1
<code>params.type</code>	string	The record type of the record being processed by the mass update.	Version 2016 Release 1

Syntax

Important: The following code snippet shows the syntax for this entry point. It is not a functional example. For a full script sample, see [Mass Update Script Sample](#).

```

// Add additional code
...
function each(params) {
    // Set the probability to 61%
    var recOpportunity = record.load({
        type: params.type,

```

```

        id: params.id
    });
    ...
    // Add additional code

```

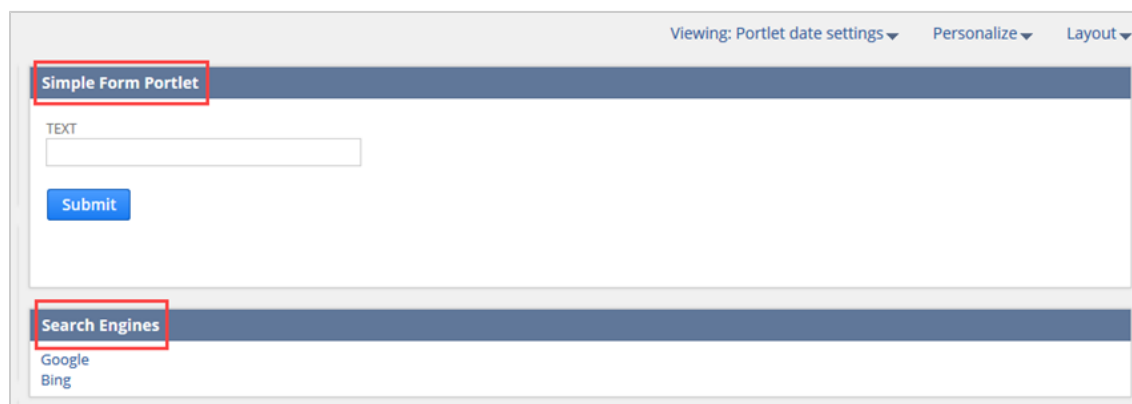
SuiteScript 2.0 Portlet Script Type

Portlet scripts are run on the server and are rendered in the NetSuite dashboard.

The following portlet script types are supported:

- **FORM** — A data entry form with up to one submit button embedded into a portlet. This type supports the Portlet module that can refresh and resize the portlet, as well as the use of record-level client-side script to implement validation. See [N/portlet Module](#).
- **HTML** — An HTML-based portlet that is used to display free-form HTML. (images, Flash, custom HTML)
- **LINKS** — A portlet that consists of rows of formatted content.
- **LIST** — A standard list of user-defined column headers and rows.

The following image shows a form portlet and a links portlet displayed on the NetSuite dashboard.



You can designate that a portlet script should be used for a SuiteApp portlet. A SuiteApp portlet is a specialized type of custom portlet. It provides direct access from users' dashboards to a SuiteApp installed in their account. This type of script is called a Dashboard SuiteApp portlet script. If a Dashboard SuiteApp portlet is included in a SuiteApp, a user can add a SuiteApp portlet to their dashboard from the Personalize Dashboard menu. For instructions, see the help topic [SuiteApp Portlets](#). This type of portlet is supported for installed SuiteApps that include a dashboard component. A SuiteApp portlet script not only provides content for SuiteApp portlets. It also enables you to include your choice of graphics as branding for the icons that are shown for SuiteApp portlets in the Personalize Dashboard window.

To view content produced by a portlet script that is not intended for a SuiteApp portlet, a user must add a custom portlet to their dashboard and select the script in the portlet setup. For more information, see the help topics [Custom Portlets](#) and [Adding a Portlet to a Dashboard](#).

For steps to create and deploy a portlet script, see [Creating and Deploying a Portlet Script](#).

See the following for more information about the Portlet Script type:

- [SuiteScript 2.0 Portlet Script Reference](#)
 - [Creating and Deploying a Portlet Script](#)

- Guidelines for Creating a Dashboard SuiteApp Icon
- SuiteScript 2.0 Portlet Script Entry Points and API
- render(params)
- Portlet Object
 - Portlet.addColumn(options)
 - Portlet.addEditColumn(options)
 - Portlet.addField(options)
 - Portlet.addLine(options)
 - Portlet.addRow(options)
 - Portlet.addRows(options)
 - Portlet.setSubmitButton(options)
 - Portlet.clientScriptFileld
 - Portlet.clientScriptModulePath
 - Portlet.html
 - Portlet.title

SuiteScript 2.0 Portlet Script Samples

- Form Portlet Script Sample
- HTML Portlet Script Sample
- Links Portlet Script Sample
- List Portlet Script Sample

Form Portlet Script Sample

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType Portlet
 */
define(['N/search'],
  function(search) {
    function render(params) {
      var portlet = params.portlet;
      portlet.title = 'Simple Form Portlet';
      var fld = portlet.addField({
        id: 'text',
        type: 'text',
        label: 'Text'
      });
      fld.updateLayoutType({
        layoutType: 'normal'
      });
      fld.updateBreakType({
        breakType: 'startcol'
      });
    }
  });
});
```

```

        portlet.setSubmitButton({
            url: 'http://httpbin.org/post',
            label: 'Submit',
            target: '_top'
        });
    }
    return {
        render: render
    };
};
});

```

HTML Portlet Script Sample

```

/**
 *@NApiVersion 2.x
 *@NScriptType Portlet
 */
define([],
    function() {
        function render(params) {
            params.portlet.title = 'My Portlet';
            var content = '<td><span><b>Hello!!!</b></span></td>';
            params.portlet.html = content;
        }
        return {
            render: render
        };
    });

```

Links Portlet Script Sample

```

/**
 *@NApiVersion 2.x
 *@NScriptType Portlet
 */
define(['N/search'],
    function(search) {
        function render(params) {
            var portlet = params.portlet;
            portlet.title = 'Search Engines';
            portlet.addLine({
                text: 'Google',
                url: 'http://www.google.com/'
            });
            portlet.addLine({
                text: 'Bing',
                url: 'http://www.bing.com/'
            });
        }
        return {
            render: render
        };
    });

```

List Portlet Script Sample



Important: This sample references a custom entity field with the ID `custentity_multiselect`. Before attempting to use this sample, you must either create a field with that ID or edit the sample so that it references an existing custom entity field in your account.

```
/**
 *@NApiVersion 2.x
 *@NScriptType Portlet
 */
define(['N/search'],
function(search) {
    function render(params) {
        var isDetail = (params.column == 2);
        var portlet = params.portlet;
        portlet.title = isDetail ? "My Detailed List" : "My List";
        portlet.addColumn({
            id: 'internalid',
            type: 'text',
            label: 'Number',
            align: 'LEFT'
        });
        portlet.addColumn({
            id: 'entityid',
            type: 'text',
            label: 'ID',
            align: 'LEFT'
        });
        if (isDetail) {
            portlet.addColumn({
                id: 'email',
                type: 'text',
                label: 'E-mail',
                align: 'LEFT'
            });
            portlet.addColumn({
                id: 'custentity_multiselect',
                type: 'text',
                label: 'Multiselect',
                align: 'LEFT'
            });
        }
        var filter = search.createFilter({
            name: 'email',
            operator: search.Operator.ISNOTEMPTY
        });
        var customerSearch = search.create({
            type: 'customer',
            filters: filter,
            columns: ['internalid', 'entityid', 'email', 'custentity_multiselect']
        });
        var count = isDetail ? 15 : 5;
        customerSearch.run().each(function(result) {
            portlet.addRow(result.getAllValues());
        });
    }
});
```

```

        return --count > 0;
    });
}
return {
    render: render
};
});

```

SuiteScript 2.0 Portlet Script Reference

- [Creating and Deploying a Portlet Script](#)
- [Guidelines for Creating a Dashboard SuiteApp Icon](#)

Creating and Deploying a Portlet Script

The following is a basic process flow for creating and deploying a portlet script, including a Dashboard SuiteApp portlet script:

1. Create a portlet script and upload the file to your File Cabinet.
For more information about this process, see [SuiteScript 2.0 Script Creation Process](#).
2. Create a script record for your portlet script, and ensure that the **Portlet Type** field is set to the correct portlet type used in your script.
For more information about this process, see [Creating a Script Record](#).
3. Create a script deployment record for your portlet script.

If you are creating a script to be used for a SuiteApp portlet, enable the **Dashboard SuiteApp** option.

When this option is enabled, you can upload an image to the file cabinet. This image represents the SuiteApp icon shown for the portlet in the Personalize Dashboard window. Note that only SVG images are supported for the icon. SVG is a vector format, so it assures perfect image scalability. For more information about icon guidelines, see [Guidelines for Creating a Dashboard SuiteApp Icon](#).

The following image indicates where the Dashboard fields are located:

The screenshot shows the 'Script Deployment' form. At the top are 'Save', 'Cancel', and 'Reset' buttons. Below is the 'SCRIPT' section with the title 'Financial Report Systems Form Portlet'. The 'TITLE' field is filled with 'Financial Report Systems Form Portlet'. The 'ID' field is empty. The 'DEPLOYED' checkbox is checked. The 'DASHBOARD SUITEAPP' checkbox is also checked and is highlighted with a red rectangular box. Below this, the 'ICON' field is set to 'NetSuite Logo' with a dropdown arrow.

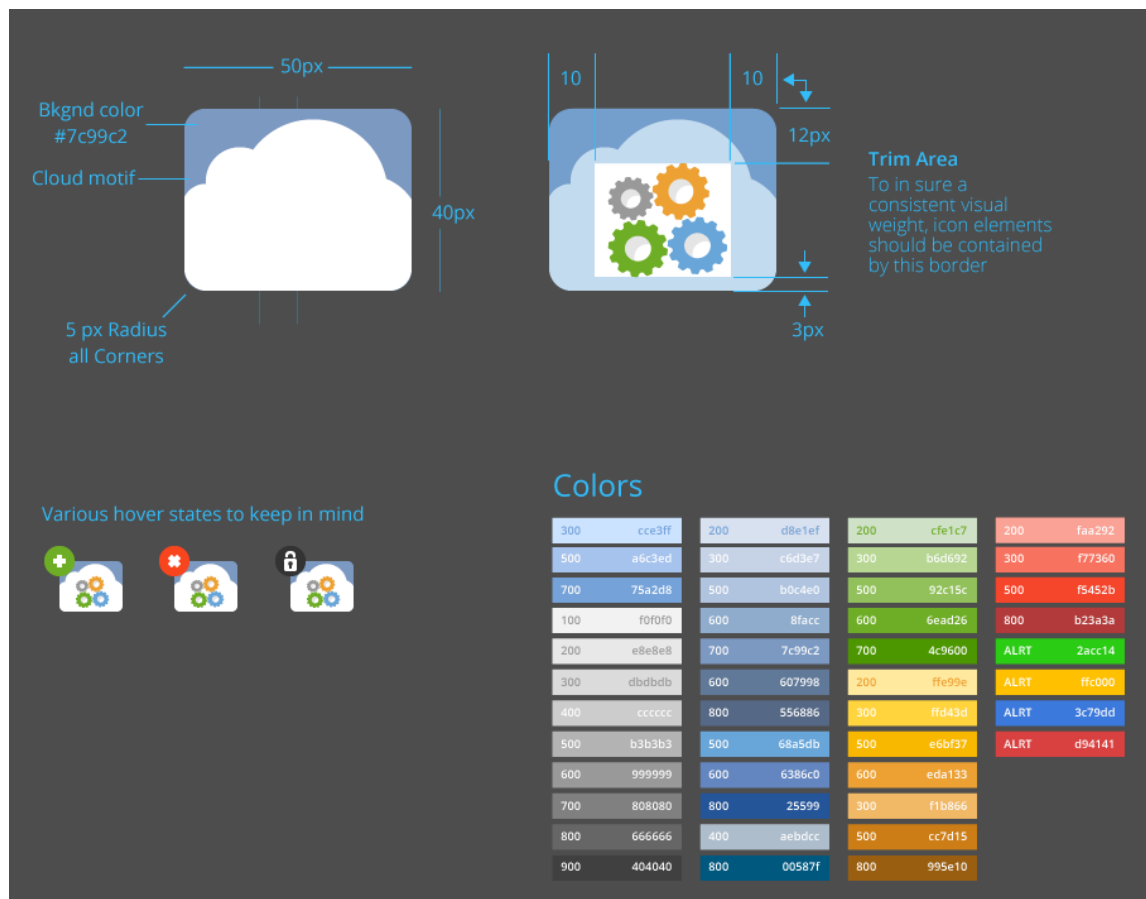
For more information about the script deployment process, see [Methods of Deploying a Script](#).

Guidelines for Creating a Dashboard SuiteApp Icon

Dashboard SuiteApp icons require certain visual characteristics to align them with other icons in the dashboard. The following guidelines preserve the visual characteristics of Dashboard SuiteApp icons:

- Dashboard SuiteApp icons use a unique background with a cloud motif. This background ensures a common size and shape for all Dashboard SuiteApp icons and provides a cloud motif that serves as an additional modifier.
- NetSuite recommends using a restricted color palette. This palette is a major contributor to the character of the icons used in NetSuite. Restrict your colors to this palette whenever possible. If additional colors are required, add them as special exceptions. See the image below for more information about the recommended color palette.
- Graphical elements should be more geometrical than illustrative. Avoid using complex and irregular shapes. Try to reduce elements in your composition to their most primitive geometry.
- Dashboard SuiteApp icons use a simulated light source to increase detail and definition in icon elements. This light source is evidenced by a cast shadow that proceeds down and to the right of elements in the composition at a 45 degree angle. Cast shadows are rendered with a black fill set to 10% opacity.
- Your icon image must be saved as an SVG file.

The following image shows the guidelines in use:



SuiteScript 2.0 Portlet Script Entry Points and API

- `render(params)`
- Portlet Object
 - `Portlet.addColumn(options)`
 - `Portlet.addEditColumn(options)`
 - `Portlet.addField(options)`
 - `Portlet.addLine(options)`
 - `Portlet.addRow(options)`
 - `Portlet.addRows(options)`
 - `Portlet.setSubmitButton(options)`
 - `Portlet.clientScriptFileId`
 - `Portlet.clientScriptModulePath`
 - `Portlet.html`
 - `Portlet.title`

render(params)

Description	Definition of the portlet script trigger point.
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `params` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>params.portlet</code>	Portlet Object	<p>The Portlet object used for rendering. The availability of Portlet members depends on the type of Portlet that is passed in. For more information, see the following:</p> <ul style="list-style-type: none"> ■ Form Portlet Object Members ■ HTML Portlet Object Members ■ Links Portlet Object Members ■ List Portlet Object Members 	Version 2015 Release 2
<code>params.column</code>	string	<p>The column index for the portlet on the dashboard. The index is a string representation of one of the following numeric values:</p> <ol style="list-style-type: none"> 1. left column 2. center column 3. right column 	Version 2015 Release 2
<code>params.entityid</code>	string	The customer ID for the selected customer.	Version 2015 Release 2

Portlet Object

Portlet objects are used to encapsulate scriptable dashboard portlets. They are automatically passed to the `render(params)` entry point by NetSuite. For more information, see [render\(params\)](#).

Form Portlet Object Members

These members are only available to Form Portlet objects.

Member Type	Name	Return Type / Value Type	Description
Method	Portlet.addField(options)	serverWidget.Field	Adds a field to the form.
	Portlet.setSubmitButton(options)	serverWidget.Button	Adds a submit button to the form.
Property	Portlet.clientScriptFileId	number	The script file ID to be used in the portlet.
	Portlet.clientScriptModulePath	string	The script path to be used in the portlet.
	Portlet.title	string	The title of the portlet.

HTML Portlet Object Members

These members are only available to HTML Portlet objects.

Member Type	Name	Return Type / Value Type	Description
Property	Portlet.html	string	The complete HTML contents of the portlet.
	Portlet.title	string	The title of the portlet.

Links Portlet Object Members

These members are only available to Links Portlet objects.

Member Type	Name	Return Type / Value Type	Description
Method	Portlet.addLine(options)	Object	Adds a line to the portlet.
Property	Portlet.title	string	The title of the portlet.

List Portlet Object Members

These members are only available to List Portlet objects.

Member Type	Name	Return Type / Value Type	Description
Method	Portlet.addColumn(options)	serverWidget.ListColumn	Adds a column to the portlet.
	Portlet.addEditColumn(options)	serverWidget.ListColumn	Adds an Edit or Edit/View column to the portlet.

Member Type	Name	Return Type / Value Type	Description
	<code>Portlet.addRow(options)</code>	Object	Adds a row to the portlet.
	<code>Portlet.addRows(options)</code>	Object	Adds multiple rows to the portlet.
Property	<code>Portlet.title</code>	string	The title of the portlet.

Portlet.addColumn(options)

Method Description	Adds a list column to the portlet.
Returns	<code>serverWidget.ListColumn</code>
Entry Point	<code>render(params)</code>
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of this column. The internal ID must be in lowercase, contain no spaces.	2016.2
<code>options.label</code>	string	required	The label of this column.	2016.2
<code>options.type</code>	string	required	The field type for this column. For more information about possible values, see <code>serverWidget.FieldType</code> .	2016.2
<code>options.align</code>	string	optional	The layout justification for this column. For more information about possible values, see <code>serverWidget.LayoutJustification</code> .	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
var newColumn = params.portlet.addColumn({
  id: 'column1',
  label: 'Text',
  type: serverWidget.FieldType.TEXT,
  align: serverWidget.LayoutJustification.RIGHT
});
...
```

Portlet.addEditColumn(options)

Method Description	Adds an Edit or Edit/View column to the portlet.
--------------------	--

Returns	<code>serverWidget.ListColumn</code>
Entry Point	<code>render(params)</code>
Since	2016.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.column</code>	string	required	The internal ID of the column to the left of which the Edit/View column is added.	2016.2
<code>options.showHrefCol</code>	boolean true false	optional	If <code>true</code> , the URL for the link is clickable. The default setting is <code>false</code> .	2016.2
<code>options.showView</code>	boolean true false	optional	If <code>true</code> , then an Edit/View column is added. Otherwise, only an Edit column is added. The default setting is <code>false</code> .	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
var newColumn = params.portlet.addEditColumn({
  column: 'column1',
  showHrefCol: true,
  showView: true
});
...
```

Portlet.addField(options)

Method Description	Adds a field to the form.
Returns	<code>serverWidget.Field</code>
Entry Point	<code>render(params)</code>
Since	2016.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of the field. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the field to an existing page. For example, if	2016.2

Parameter	Type	Required / Optional	Description	Since
			you add a field that appears as Purchase Details , the field internal ID should be something similar to <code>custpage_purchasedetails</code> or <code>custpage_purchase_details</code> .	
<code>options.label</code>	string	required	The label for this field.	2016.2
<code>options.type</code>	string	required	The field type for the field. For more information about possible values, see serverWidget.FieldType .	2016.2
<code>options.source</code>	string	optional	The internal ID or script ID of the source list for this field if it is a select (List/Record) or multi-select field. <div> <p>Note: For radio fields only, the source parameter must contain the internal ID for the field.</p> <p>Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with <code>Field.addSelectOption(options)</code>. The select values are determined by the source record or list.</p> </div> <p>For more information about working with radio buttons, see the help topic Working with Radio Buttons.</p>	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
var newField = params.portlet.addField({
  id: 'textfield',
  type: serverWidget.FieldType.TEXT,
  label: 'text'
});
...
```

Portlet.addLine(options)

Method Description	Adds a line to the portlet.
Returns	Object
Entry Point	render(params)
Since	2016.2

Parameters

Note:	The options parameter is a JavaScript object.
--------------	---

Parameter	Type	Required / Optional	Description	Since
<code>options.text</code>	string	required	The text for the line.	2016.2

Parameter	Type	Required / Optional	Description	Since
options.url	string	optional	The URL link.	2016.2
options.align	number	optional	The justification for the line. This value indicates the number of spaces to indent.	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.addLine({
  text: 'Google',
  url: 'http://www.google.com',
  align: 4
});
...
```

Portlet.addRow(options)

Method Description	Adds a row to the portlet.
Returns	Object
Entry Point	render(params)
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.row	search.Result string	required	A row that consists of either a search.Result , or name/value pairs. Each pair should contain the value for the corresponding Column object in the list.	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.addRow({
  row: {
    columnid1: 'value1',
    columnid2: 'value2'
```

```

    }
  });
  ...

```

Portlet.addRows(options)

Method Description	Adds multiple rows to the portlet.
Returns	Object
Entry Point	render(params)
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.rows	search.Result[] string[]	required	An array of rows that consist of either a search.Result array, or an array of name/value pairs. Each pair should contain the value for the corresponding Column object in the list.	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```

...
params.portlet.addRows({
  rows:
    [{
      columnid1: 'value1',
      columnid2: 'value2'
    }, {
      columnid1: 'value2',
      columnid2: 'value3'
    }]
});
...

```

Portlet.setSubmitButton(options)

Method Description	Adds a submit button to the form.
Returns	serverWidget.Button
Entry Point	render(params)
Since	2016.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The URL that the form posts data to.	2016.2
options.label	string	optional	The button label.	2016.2
options.target	string	optional	The target attribute of the form element, if it is different from the portlet's own embedded iframe. Supported values include standard HTML target attributes such as <code>_top</code> , <code>_parent</code> , and <code>_blank</code> . It can also be set to frame names and the NetSuite-specific identifier <code>_hidden</code> . Setting this value to <code>_hidden</code> allows submission to a backend that returns results to a hidden child iframe within the portlet's embedded iframe.	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.setSubmitButton({
  url: 'http://httpbin.org/post',
  label: 'Submit',
  target: '_top'
});
...
```

Portlet.clientScriptFileId

Property Description	The script file ID to be used in the portlet.
Type	number
Entry Point	render(params)
Since	2016.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the <code>Portlet.clientScriptModulePath</code> property value has already been specified. For more information, see Portlet.clientScriptFileId .

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.clientScriptFileId = 32;
```

...

Portlet.clientScriptModulePath

Property Description	The script path to be used in the portlet.
Type	string
Entry Point	render(params)
Since	2016.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the Portlet.clientScriptFileId property value has already been specified. For more information, see Portlet.clientScriptFileId .

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.clientScriptModulePath = '/SuiteScripts/clientScript.js';
...
```

Portlet.html

Property Description	The complete HTML contents of the portlet.
Type	string
Entry Point	render(params)
Since	2016.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.html = htmlcontents;
...
```

Portlet.title

Property Description	The title of the portlet.
Type	string
Entry Point	render(params)

Since	2016.2
-------	--------

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [SuiteScript 2.0 Portlet Script Type](#).

```
...
params.portlet.title = 'My Portlet';
...
```

SuiteScript 2.0 RESTlet Script Type

A RESTlet is a SuiteScript that you make available for other applications to call, either from an external application or from within NetSuite. When an application or another script calls a RESTlet, the RESTlet script executes and, in some cases, returns a value to the calling application.

RESTlets can be useful when you want to bring data from another system into NetSuite, or if you want to extract data from NetSuite. RESTlets can also be used, in combination with other scripts, to customize the behavior of a page within NetSuite.

For more details, see the following sections:

- [SuiteScript 2.0 Getting Started with RESTlets](#)
- [RESTlet Authentication](#)
- [SuiteScript 2.0 RESTlet Reference](#)
- [SuiteScript 2.0 RESTlet Script and Request Samples](#)
- [SuiteScript 2.0 RESTlet Script Entry Points](#)

SuiteScript 2.0 Getting Started with RESTlets

For help getting started with RESTlets, see the following topics:

- [RESTlet Key Concepts](#)
- [Deploying a RESTlet](#)
- [Identifying a RESTlet in a Call](#)
- [Selecting an HTTP Method for Calling a RESTlet](#)
- [Creating a Content-Type Header](#)

RESTlet Key Concepts

A RESTlet is a SuiteScript that executes when called by an external application or by another SuiteScript. Depending on how the RESTlet is written and called, it may also return data to the calling application.

As with other script types, RESTlets have broad potential applications. A RESTlet can perform any function that can be implemented by using SuiteScript. But at a high level, potential uses of RESTlets generally include the following:

- Retrieving, adding, or manipulating data within NetSuite, from an external source. In this sense, RESTlets can be seen as an alternative to NetSuite's SOAP-based web services.

- Customizing the behavior of pages and features within NetSuite. In this sense, RESTlets can be seen an alternative to other script types, such as server-side Suitelets. The advantage of using a RESTlet compared with a Suitelet is that the RESTlet can return data, in plain text or JSON, to the client script.

To use a RESTlet, you follow the same guidelines as you would with an entry point script deployed at the record level. For example, you must create a script record and a deployment record based on the RESTlet script file. These processes are described further in [Deploying a RESTlet](#).

When you save a script deployment record for a RESTlet, the system automatically generates a URL that can be used to call the RESTlet. Because a RESTlet executes only when it is called, this information is critical for using the RESTlet. For more details, see [Identifying a RESTlet in a Call](#).

When you are ready to call a RESTlet that you have deployed, you can use one of four supported HTTP methods: delete, get, post, or put. Depending on which method you use, you may be required to embed input for the RESTlet in the URL, or you may be required to submit arguments in a request body. Additionally, for the call to be successful, your RESTlet script must contain an entry point that corresponds with the method you use to make the call. For details on supported HTTP methods and formatting your request, see [Selecting an HTTP Method for Calling a RESTlet](#).

One advantage of RESTlets over other script types is that NetSuite requires authentication for RESTlets. If a RESTlet call originates from a client that does not have an existing session in the NetSuite account where the RESTlet is deployed, NetSuite requires the call to include an authorization header. For details about adding an authentication header to a RESTlet, see the help topic [RESTlet Authentication](#).

For most RESTlet calls, you must also include a content-type header, which tells NetSuite how your request body will be formatted and how NetSuite should format its response. For details, see [Creating a Content-Type Header](#).

Deploying a RESTlet

Before you can use a RESTlet, you must follow the same guidelines as you would with most entry point scripts. At a high level, you must do the following:

- [Make Sure the Script Is Formatted Properly](#)
- [Create a Script and Script Deployment Record](#)

Make Sure the Script Is Formatted Properly

All of the following must be true:

- The script must have the correct structure for SuiteScript 2.0. For example, the script must have an interface that includes at least one entry point appropriate for the RESTlet script type. The script must also contain a corresponding entry point function. For details on how to structure a SuiteScript 2.0 script, see [Correct Structure for Entry Point Scripts](#). Note that the entry points you use will determine how the RESTlet can be called. For details on the RESTlet entry points, see [SuiteScript 2.0 RESTlet Script Entry Points](#).
- The script must use the required JSDoc tags. The `@NScriptType` must be `RESTlet` (or `Restlet`; these values are not case-sensitive). For further details on the required JSDoc tags, see [Required JSDoc Tags for Entry Point Scripts](#).

Create a Script and Script Deployment Record

Before you can use a RESTlet, you must upload it to your File Cabinet. Then you use the file to create a script record and a script deployment record.

For general help creating script records and script deployment records, see [SuiteScript 2.0 Record-Level Scripts](#).

When creating these records for a RESTlet, be aware of the following:

- NetSuite recommends that you enter meaningful data in the script record's ID field and the script deployment record's ID field. When you save the records, the system creates IDs that include the text you entered. One possible use of these IDs is to identify the RESTlet when calling it from another SuiteScript. For this reason, it may be helpful to have created meaningful ID strings.
- Unlike some other script types, you do not deploy a RESTlet for any particular record type. Each RESTlet is available independently of any particular record type or record instance.
- The script deployment record includes a field called Status, which has possible values of Released and Testing. Before you can call the RESTlet from an external source, the Status field must be set to Released.
- When you save a script deployment record for a RESTlet, the system automatically generates a partial and full URL that you can use to call the RESTlet. However, if you are calling the RESTlet from within an integration and you want to use the full URL, you must include logic that dynamically discovers the RESTlet domain. See the following image for an example of internal and external RESTlet URLs. For more information, see [Identifying a RESTlet in a Call](#).

Script Deployment

[Edit](#) [Back](#) [Actions](#)

SCRIPT Advanced Promotions Client Translator	STATUS Testing
TITLE Advanced Promotions Client Translator 2	LOG LEVEL Debug
ID customdeploy2	URL /app/site/hosting/restlet.nl?script=58&deploy=2
☒ DEPLOYED	EXTERNAL URL https://123456.restlets.apl.netsuite.com/app/site/hosting/restlet.nl?script=58&deploy=2

Identifying a RESTlet in a Call

Before you can access a RESTlet, you must know how to identify it in your call. In general, you use values from the script deployment record. In some cases, you may also have to use the ID value from the script record.

Script

[Edit](#) [Back](#) [Deploy Script](#) [Actions](#)

TYPE
RESTlet

NAME
Process Customer Data

1 ID
customscript_processcustomerdata

API VERSION
2.0

Script Deployment

[Edit](#) [Back](#) [Actions](#)

SCRIPT
Process Customer Data

TITLE
_processcustomerdata

2 ID
customdeploy_processcustomerdata

☒ DEPLOYED

3 URL
/app/site/hosting/restlet.nl?script=478&deploy=1

4 EXTERNAL URL
https://rest.netsuite.com/app/site/hosting/restlet.nl?script=478&deploy=1

For details, see the following sections:

- [Internal Versus External Calls](#)
- [Dynamically Generating a Full URL](#)

Internal Versus External Calls

The values you use to identify a RESTlet vary depending on the source of the call. For details, see the following table. Each number in the table refers to a callout in the screenshot above.

Source of the Call	Identify the RESTlet with:
An external client	A full URL, similar to the one shown on the script deployment record, in the External URL field (4). For details, see Dynamically Generating a Full URL .
A client SuiteScript with an active session in the same NetSuite account where the RESTlet is deployed	Either of the following: <ul style="list-style-type: none"> ■ The partial URL shown on the script deployment record, in the URL field (3). For an example, see Example: Client Script that Calls a RESTlet. ■ A URL generated by using the N/url module. To generate the URL this way, you need the script ID, which is viewable on the script record in the ID field (1), in combination with the deployment ID, which is viewable on the script deployment record, in its ID field (2). For an example, see Example: Suitelet that Calls a RESTlet.
A server SuiteScript in the same NetSuite account where the RESTlet is deployed	A full URL, similar to the one shown on the script deployment record, in the External URL field (4). This URL must be dynamically generated, in one of the following ways: <ul style="list-style-type: none"> ■ By using the REST roles service. For details, see the help topic The REST roles Service. ■ By using the N/url module. To generate the URL this way, you need the script ID, which is viewable on the script record in the ID field (1), in combination with the deployment ID, which is viewable on the script deployment record, in its ID field (2). For an example, see Example: Suitelet that Calls a RESTlet.
A server SuiteScript in a different NetSuite account from where the RESTlet is deployed	A full URL, similar to the one shown on the script deployment record, in the External URL field (4). This URL must be dynamically generated by using the REST roles service. For details, see Dynamically Generating a Full URL .

Note: If you are calling a RESTlet by using the delete or get method, you must extend the URL to include any input data that is required by the RESTlet's logic. For details, see [Selecting an HTTP Method for Calling a RESTlet](#).

Dynamically Generating a Full URL

When you save a script deployment record for a RESTlet, the system automatically generates a full URL that can be used to call the RESTlet. This value is shown in the **External URL** field.

However, in general, you should not hard-code this URL in a script, or in any other part of your integration. Instead, you should create logic that dynamically generates the portion of the URL that represents the RESTlet domain.

The RESTlet domain is the first part of the URL. It appears as `https://rest.netsuite.com`, `https://rest.na1.netsuite.com`, or a similar variant. You must dynamically generate this portion of the URL because it can change. The domain can change because NetSuite hosts accounts in multiple data centers, and accounts are sometimes moved. The External URL value on the script deployment record is correct and will be updated if your account moves. Your integration must also be adaptable.

Use the following approaches:

- For calling a RESTlet from an external source, use NetSuite's roles service. For details on using this service, see the help topic [The REST roles Service](#).
- If you are calling a RESTlet from within NetSuite, you can use the N/url module. With this approach, you provide the ID values from the script record and script deployment record. For an example, see [Example: Suitelet that Calls a RESTlet](#).

Note: As of 2017.2, account-specific domains are supported for RESTlets, and you can access your RESTlet domain at the following URL, where 123456 is your account ID: 123456.restlets.api.netsuite.com. The data center-specific domains supported before 2017.2 will continue to be supported. For more information, see the help topic [URLs for Account-Specific Domains](#).

Selecting an HTTP Method for Calling a RESTlet

When you call a RESTlet, you can use one of four supported HTTP methods: delete, get, post, or put.

For more details on using these methods with RESTlets, see the following sections:

- [A Call's Method Must Match an Entry Point](#)
- [HTTP Method Functionality](#)
- [Input Data Is Handled Differently by Different Methods](#)

A Call's Method Must Match an Entry Point

For each supported HTTP method there is a corresponding supported entry point. For a call to be successful, the method used in the call must match an entry point defined in the RESTlet's interface.

The snippets shown in the following screenshot include a successful pairing of a RESTlet and a call to that RESTlet: In the first snippet, the Suitelet calls the RESTlet by using the get method. Because the RESTlet's interface includes a get entry point, as shown in the second snippet, the call would be successful.

```
// Here, the Suitelet calls a RESTlet
var response = https.get({url: url, headers: headers});

// In this function, the RESTlet retrieves
// a standard NetSuite record.

function _get(context) {
    doValidation([context.recordtype, context.id], ['recordtype', 'id'], 'GET');
    return record.load({
        type: context.recordtype,
        id: context.id
    });
}

return {
    get : _get,
};
```

HTTP Method Functionality

The following HTTP methods are supported: delete, get, post, and put. These methods are defined at the following link:

<https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

However, be aware that the way a RESTlet behaves is defined in the RESTlet script and may not necessarily correlate with the intended behavior of the HTTP method being used. Although it is not

recommended, your RESTlet could for example define an entry point get function that creates data, rather than retrieving it. Similarly, whether or not a method requires data varies depending on how you write the function. You could write a post entry point function that does not require input.

Input Data Is Handled Differently by Different Methods

The way you pass input parameters to a RESTlet varies depending on the HTTP method you use, as described in the following table.

Method	Placement of required input
delete	Arguments must be embedded in the URL used to make the call.
get	
post	Arguments must be included in a request body written in JSON (JavaScript Object Notation) or plain text.
put	

Note: For examples of requests, see [SuiteScript 2.0 RESTlet Script and Request Samples](#).

Creating a Content-Type Header

Depending on the design of a RESTlet, calls to that RESTlet may require a request body. Additionally, sometimes data is returned by a RESTlet, and you may want to use that data.

In the first situation, you must tell NetSuite how your request body will be formatted. In the second, you might want to specify the format of the data that is returned. Note that both formats must be the same. You control this choice by adding a content-type header to your request.

For details on the supported values for this header, see the following table.

Value	Notes
application/json	JSON is the appropriate choice for most RESTlets that require a request body, because it lets you map values to fields.
application/xml	XML is supported only for the get method.
text/plain	Because plain text does not let you map values to fields, this choice should be used only for RESTlets that require limited and simple input.

Structuring the Header

Format your content-type header as follows:

```
Content-Type: application/json
```

Note that content-type header values are case-sensitive.

For an example of a shell script that generates a content-type header, see [Example: Shell Script that Calls a RESTlet](#).

Error Handling

If you omit a content-type header, the request fails with an HTTP error code reading 206: Partial Content.

SuiteScript 2.0 RESTlet Reference

See the following topics for more details about working with RESTlets:

- [SuiteScript 2.0 RESTlet Error Handling](#)
- [SuiteScript 2.0 RESTlet Governance](#)
- [SuiteScript 2.0 RESTlet Security](#)

SuiteScript 2.0 RESTlet Error Handling

For details about error handling for RESTlets, refer to the following sections:

- [Supported HTTP Status Codes for RESTlets](#)
- [SuiteScript Errors Returned by RESTlets](#)

Supported HTTP Status Codes for RESTlets

For details about the HTTP status codes supported for RESTlets, see:

- [HTTP Success Code](#)
- [HTTP Error Codes](#)

HTTP Success Code

NetSuite supports one HTTP success code for RESTlets: **200 OK**. This code indicates that the request was executed successfully. Note that this code does not necessarily mean that your request worked as you intended. In some cases, a SuiteScript error might occur and be successfully handled by the RESTlet script. In these cases, an HTTP code of 200 is used, and details of the error are described in the response body.

HTTP Error Codes

NetSuite supports the following HTTP error codes for RESTlets.

Code	Description
206 Partial Content	You receive this error if you attempt to use a content-type that is unsupported for the HTTP method you are using.
302 Moved Temporarily	This error can be displayed if your request went to the wrong NetSuite data center. Review your integration and make sure that it generates the NetSuite RESTlet domain dynamically. For details, see Dynamically Generating a Full URL .
400 Bad Request	The RESTlet request failed with a user error. Any errors encountered at run time that are unhandled return a 400 error. (If the user code catches the error, a status code of 200 is returned.)
401 Unauthorized	There is no valid NetSuite login session for the RESTlet call.
403 Forbidden	The RESTlet request was sent to an invalid domain, meaning a domain other than <code>https://rest.netsuite.com</code> , <code>https://rest.na1.netsuite.com</code> , or a similar variant.
404 Not Found	A RESTlet script is not defined in the RESTlet request.
405 Method Not Allowed	The request method used is not valid.

Code	Description
415 Unsupported Media Type	An unsupported content type was specified.
500 Internal Server Error	This error occurs for non-user errors that cannot be recovered by resubmitting the same request. If this type of error occurs, contact Customer Support to file a case.
503 Service Unavailable	The NetSuite database is offline, or a database connection is not available.

SuiteScript Errors Returned by RESTlets

In some cases, the response to a RESTlet is a SuiteScript error. For details, see the following sections:

- [SuiteScript Error Codes Used by RESTlets](#)
- [SuiteScript Error Message Formatting for RESTlets](#)

SuiteScript Error Codes Used by RESTlets

The following table describes some of the supported SuiteScript errors that can occur when using RESTlets.

Code	Description	Notes
INVALID_LOGIN_ATTEMPT	Invalid login attempt.	This error indicates a problem in an OAuth header. It can be returned when the nonce, consumer key, token, or signature in the OAuth header is invalid.
INVALID_LOGIN_CREDENTIALS	You have entered an invalid email address or account number. Please try again.	This error indicates a problem in an OAuth header.
INVALID_REQUEST	The request could not be understood by the server due to malformed syntax.	This error is returned because of malformed syntax in an OAuth header. For example, this error can occur when the signature method, version, or timestamp parameter is rejected.
INVALID_RETURN_DATA_FORMAT	You should return {1}.	This error is used if the response data does not match the expected format, as specified by the content-type header.
SSS_INVALID_SCRIPTLET_ID	That Suitelet is invalid, disabled, or no longer exists.	If you receive this error, make sure that the URL points to the correct RESTlet script deployment ID.
UNEXPECTED_ERROR	An unexpected error occurred. Error ID: {1}	

SuiteScript Error Message Formatting for RESTlets

The following examples illustrate SuiteScript errors formatting for each supported content-type.

JSON

```
{
  "error":
  {
```



```

    "code": "SSS_INVALID_SCRIPTLET_ID",
    "message": "That Suitelet is invalid, disabled, or no longer exists."
  }
}

```

XML

```

<error>
  <code>SSS_INVALID_SCRIPTLET_ID</code>
  <message>That Suitelet is invalid, disabled, or no longer exists.</message>
</error>

```

Text

```

error code: SSS_INVALID_SCRIPTLET_ID
error message: That Suitelet is invalid, disabled, or no longer exists.

```

SuiteScript 2.0 RESTlet Governance

The SuiteScript governance model tracks usage units on two levels: API level and script level. At the API level, RESTlets have the same usage limits as other types of SuiteScripts. At the script level, RESTlets allow 5,000 usage units per script, a limit five times greater than Suitelets and most other types of SuiteScripts. For more information, see the help topic [SuiteScript Governance](#).

There is a limit of 10MB per string used as RESTlet input or output.

SuiteScript currently does not support a logout operation similar to the one used to terminate a session in SuiteTalk.



Important: Starting from version 2017.2, web services and RESTlet concurrency is governed per account. The new account governance limit applies to the combined total of web services and RESTlet requests. For details about this change, see the help topic [Web Services and RESTlet Concurrency Governance](#).

SuiteScript 2.0 RESTlet Security

The URLs for accessing RESTlets are protected by TLS encryption. Only requests sent using TLS encryption are granted access. For more information on supported versions of the TLS protocol, see the help topic [FAQ: Transport Layer Security \(TLS\) Deprecations](#).

SuiteScript 2.0 RESTlet Script and Request Samples

For sample RESTlet scripts and requests, see the following sections:

- [Example: Hello World RESTlet](#)
- [Example: RESTlet that Can Retrieve, Delete or Create](#)
- [Example: RESTlet that Adds Multiple Records](#)
- [Example: RESTlet that Manipulates Scheduled Script](#)
- [Example: Client Script that Calls a RESTlet](#)
- [Example: Suitelet that Calls a RESTlet](#)

■ Example: Shell Script that Calls a RESTlet

For help with writing other types of scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Example: Hello World RESTlet

Using RESTlets can be more complicated than using other script types because, whereas other script types that can be deployed and then will execute as needed, a RESTlet must be deployed and then called. To call a RESTlet successfully, you must correctly identify your NetSuite account's RESTlet domain, use an HTTP method that matches an entry point in your script, and use two required headers: Content-Type and Authorization. (For details on these headers, see [Creating a Content-Type Header](#) and [RESTlet Authentication](#).)

When getting started, you may want to test your RESTlet setup with a simple script such as the following.

RESTlet

```
/**
 * @NApiVersion 2.x
 * @NScriptType restlet
 */
define([], function() {
  return {
    get : function() {
      return "Hello World!"
    }
  }
});
```

Sample Get Call and Response

You call this RESTlet by using the get method. Because the RESTlet takes no arguments, you would not need to extend the URL with additional values. For testing purposes, you could use the value that appears in the External URL field of the script deployment record. (However, in an integration, remember that you must dynamically discover the RESTlet domain.)

For example, you could call this RESTlet by using a URL like the following — one that does not include embedded parameters:

```
https://rest.netsuite.com/app/site/hosting/restlet.nl?script=482&deploy=1
```

After you add the two required headers, the generated call would look like the following.

```
GET /app/site/hosting/restlet.nl?script=482&deploy=1 HTTP/1.1
HOST: rest.netsuite.com
authorization: NLAUTH nlauth_account=12345, nlauth_email=john@smith.com, nlauth_signature=Welcome123
content-type: application/json
cookie: ...
```

The RESTlet would return the following response:

```
Hello World!
```

Example: RESTlet that Can Retrieve, Delete or Create

The following RESTlet includes logic for all supported entry points: get, delete, post, and put.

RESTlet

```
/**
 * @NApiVersion 2.x
 * @NScriptType Restlet
 */
define(['N/record', 'N/error'],
function(record, error) {
    function doValidation(args, argNames, methodName) {
        for (var i = 0; i < args.length; i++)
            if (!args[i] && args[i] !== 0)
                throw error.create({
                    name: 'MISSING_REQ_ARG',
                    message: 'Missing a required argument: [' + argNames[i] + '] for method: ' + methodName
                });
    }

    // Get a standard NetSuite record
    function _get(context) {
        doValidation([context.recordtype, context.id], ['recordtype', 'id'], 'GET');
        return JSON.stringify(record.load({
            type: context.recordtype,
            id: context.id
        }));
    }

    // Delete a standard NetSuite record
    function _delete(context) {
        doValidation([context.recordtype, context.id], ['recordtype', 'id'], 'DELETE');
        record.delete({
            type: context.recordtype,
            id: context.id
        });
        return String(context.id);
    }

    // Create a NetSuite record from request params
    function post(context) {
        doValidation([context.recordtype], ['recordtype'], 'POST');
        var rec = record.create({
            type: context.recordtype
        });
        for (var fldName in context)
            if (context.hasOwnProperty(fldName))
                if (fldName !== 'recordtype')
                    rec.setValue(fldName, context[fldName]);
        var recordId = rec.save();
        return String(recordId);
    }

    // Upsert a NetSuite record from request param
    function put(context) {
        doValidation([context.recordtype, context.id], ['recordtype', 'id'], 'PUT');
        var rec = record.load({
            type: context.recordtype,
```

```

        id: context.id
    });
    for (var fldName in context)
        if (context.hasOwnProperty(fldName))
            if (fldName !== 'recordtype' && fldName !== 'id')
                rec.setValue(fldName, context[fldName]);
    rec.save();
    return JSON.stringify(rec);
}
return {
    get: _get,
    delete: _delete,
    post: post,
    put: put
};
});

```

Sample Get Call and Response

To retrieve a record by using this RESTlet, you would use the get method. To identify the record you want to retrieve, you would add values to the URL you use to call the RESTlet. These values would identify the record type and internal ID of the record instance you want. These parameters are defined in the RESTlet's get function as:

- recordtype
- id

You add a value for each parameter by using an ampersand, the name of the parameter, an equals sign, and the parameter's value, as follows:

```
&[name of parameter]=[value]
```

For example, to retrieve a phone call record with the internal ID of 9363, you would use URL like the following:

```
https://rest.netsuite.com/app/site/hosting/restlet.nl?script=474&deploy=1&recordtype=phonecall&id=9363
```

Using the get method in conjunction with this URL would produce the following request:

```

GET /app/site/hosting/restlet.nl?script=474&deploy=1&recordtype=phonecall&id=9363 HTTP/1.1
HOST: rest.netsuite.com
authorization: NLAAuth nlauth_account=12345, nlauth_email=john@smith.com, nlauth_signature=Welcome123
content-type: application/json
cookie: ...

```

In response, the system would return data like the following:

```

{
  "id": "9363"
  "type": "phonecall"
  "isDynamic": false
  "fields": {
    "winstances": ""
    "nllloc": "0"
    "nlsb": "1"
  }
}

```

```

"createddate": "5/18/2016 1:01 am"
"timezone": "America/Los_Angeles"
"accesslevel": "F"
"_eml_nkey_": "143659438"
"starttime": ""
"startdate": "5/18/2016"
"title": "Project kickoff"
"type": "call"
...
}

"sublists": {
  "contact": {
    "currentline": {
      "company": ""
      "contact": ""
      ...
    }-
  }-
}-
}

```

Sample Post Call and Response

To use this RESTlet to add a record, you would use the post method. Your arguments must be included in a request body, and the request body would have to be written in JSON rather than plain text. For example:

```
{ "recordtype": "phonecall", "type": "phonecall", "title": "Project Kickoff" }
```

You would send your post method to a URL that has not been extended. For testing purposes, you could use the value that appears in the External URL field of the script deployment record. (However, in an integration, remember that you must dynamically discover the RESTlet domain.)

For example, you could call this RESTlet by using a URL like the following — one that does not include embedded parameters:

```
https://rest.netsuite.com/app/site/hosting/restlet.nl?script=474&deploy=1
```

Making a post call using the request body above, plus the appropriate headers, would produce the following request:

```

POST /app/site/hosting/restlet.nl?script=474&deploy=1 HTTP/1.1
HOST: rest.netsuite.com
authorization: NLAAuth nlauth_account=12345, nlauth_email=john@smith.com, nlauth_signature=Welcomel23
content-type: application/json
cookie: ...

```

In response, the system returns the internal ID of the newly created record. For example:

```
9564
```

Example: RESTlet that Adds Multiple Records

The following RESTlet example can create several contact records in one call.

RESTlet

```
/**
 * @NApiVersion 2.x
 * @NScriptType restlet
 */
define([ 'N/record' ], function(record) {
    return {
        post : function(restletBody) {
            var restletData = restletBody.data;
            for ( var contact in restletData) {
                var objRecord = record.create({
                    type : record.Type.CONTACT,
                    isDynamic : true
                });
                var contactData = restletData[contact];
                for ( var key in contactData) {
                    if (contactData.hasOwnProperty(key)) {
                        objRecord.setValue({
                            fieldId : key,
                            value : contactData[key]
                        });
                    }
                }
                var recordId = objRecord.save({
                    enableSourcing : false,
                    ignoreMandatoryFields : false
                });
                log.debug(recordId);
            }
        }
    };
});
```

Sample Post Call

To create records with this RESTlet, you would use the post method. You would use a request body to send your values for the new contact records.

```
{
  "data": [
    {
      "firstname": "John",
      "middlename": "Robert",
      "lastname": "Smith",
      "subsidiary": "1"
    },
    {
      "firstname": "Anne",
      "middlename": "Doe",
      "lastname": "Smith",
      "subsidiary": "1"
    }
  ]
}
```

Example: RESTlet that Manipulates Scheduled Script

The following RESTlet passes a value to a scheduled script.

Note: The scriptId and deploymentId in this sample are placeholders. Before using this script, replace the IDs with valid values from your NetSuite account.

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

```

define([ 'N/task' ], function(task) {
    return {
        get : function() {
            var mrTask = task.create({
                taskType : task.TaskType.SCHEDULED_SCRIPT
            });
            mrTask.scriptId = 488;
            mrTask.deploymentId = 'customdeploy_scheduledscript';
            mrTask.params = {
                custscriptcustom_data : 'data'
            };
            mrTask.submit();
            return "Scheduled script updated";
        }
    };
});

```

Example: Client Script that Calls a RESTlet

The following example shows how a client script can call a RESTlet. Because the client script is expected to have an active session, it uses a partial URL to call the RESTlet. That is, the URL does not have to include the RESTlet domain. You can find this partial URL in the **URL** field of the script deployment record for the RESTlet.



Note: The partial URL in this sample is a placeholder. Before using this script, replace this string with a valid value from your NetSuite account.

```

/**
 * @NApiVersion 2.x
 * @NSScriptType ClientScript
 */
define(['N/https'], function(https) {
    return {
        pageInit : function() {
            var dataFromRestlet = https
                .get('/app/site/hosting/restlet.nl?script=200&deploy=1');
            console.log(dataFromRestlet.body);
        }
    };
});

```

Example: Suitelet that Calls a RESTlet

The following example shows how a Suitelet can call a RESTlet. Because the Suitelet is deployed in the same NetSuite account as the RESTlet, the script can use the `N/url` module to resolve the URL for the RESTlet. With this approach, you need the script ID and the script deployment ID associated with the RESTlet.



Note: The `scriptId`, `deploymentId`, and authorization data in this sample are placeholders. Before using this script, replace these values with valid data from your NetSuite account.

```

/**

```

```

* @NApiVersion 2.x
* @NScriptType Suitelet
*/
define(['N/https', 'N/url'], function(https, urlMod) {
    return {
        onRequest : function(options) {
            var url = urlMod.resolveScript({
                scriptId: 'customscript196',
                deploymentId: 'customdeploy1',
                returnExternalUrl: true,
                params: {parametername: 'value'}
            });

            var headers = {'Authorization': 'NLAAuth nlauth_account=12345, nlauth_email=john@smith.com,
nlauth_signature=Welcomel23, nlauth_role=3'};

            var response = https.get({url: url, headers: headers});

            options.response.write(response.body);
        }
    };
});

```

Example: Shell Script that Calls a RESTlet

The following shell script shows how you to send a request using NLAAuth authentication and a content-type value of application/json.

```

#!/bin/sh

#Put your RESTlet url here.
my_url="https://123456.restlets.api.netsuite.com/app/site/hosting/restlet.nl?script=126&deploy=1"

#Put the body of your request in a file named data.json.
DATA_FILE="@data.json"

CONTENT_FLAG="Content-Type: application/json"

#Update the following line with valid values from your NetSuite account.
AUTH_STRING="Authorization: NLAAuth nlauth_account=123456, nlauth_email=jsmith%40ABC.com, nlauth_signature=xxxx"

#Capture the response from your RESTlet.
/usr/bin/curl -H "${CONTENT_FLAG}" -H "${AUTH_STRING}" -d "${DATA_FILE}" $my_url > /tmp/restlet_response.txt

```

SuiteScript 2.0 RESTlet Script Entry Points

Script Entry Point	
get	All RESTlet entry points return the HTTP response body.
delete	
put	

Script Entry Point	
post	

delete

Description	Returns the HTTP response body. <ul style="list-style-type: none"> ■ Returns a string when request Content-Type is 'text/plain'. ■ Returns an Object when request Content-Type is 'application/json'.
Returns	string object
Since	Version 2015 Release 2

Parameters

 Note: The <code>requestParams</code> parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Required / Optional	Description	Since
<code>requestParams</code>	Object	required	The parameters from the HTTP request URL. For all content types, parameters are passed into the function as a JavaScript Object.	Version 2015 Release 2

get

Description	Returns the HTTP response body. <ul style="list-style-type: none"> ■ Returns a string when request Content-Type is 'text/plain'. ■ Returns an Object when request Content-Type is 'application/json' or 'application/xml'.
Returns	string object
Since	Version 2015 Release 2

Parameters

 Note: The <code>requestParams</code> parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Description	Since
<code>requestParams</code>	Object	The parameters from the HTTP request URL. For all content types, parameters are passed into the function as a JavaScript Object.	Version 2015 Release 2

post

Description	Returns the HTTP response body.
--------------------	---------------------------------

	<ul style="list-style-type: none"> ■ Returns a string when request Content-Type is 'text/plain'. ■ Returns an Object when request Content-Type is 'application/json'.
Returns	string object
Since	Version 2015 Release 2

Parameters

Note: The `requestBody` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Required / Optional	Description	Since
<code>requestBody</code>	string Object	required	The HTTP request body. <ul style="list-style-type: none"> ■ Pass the request body as a string when the request Content-Type is 'text/plain' ■ Pass the request body as a JavaScript Object when the request Content-Type is 'application/json'. 	Version 2015 Release 2

put

Description	Returns the HTTP response body. <ul style="list-style-type: none"> ■ Returns a string when request Content-Type is 'text/plain'. ■ Returns an Object when request Content-Type is 'application/json'. 			
Returns	string object			
Since	Version 2015 Release 2			

Parameters

Note: The `requestBody` parameter is a JavaScript object. It is automatically passed to the script entry point by NetSuite.

Parameter	Type	Required / Optional	Description	Since
<code>requestBody</code>	string Object	required	The HTTP request body. <ul style="list-style-type: none"> ■ Pass the request body as a string when the request Content-Type is 'text/plain' ■ Pass the request body as a JavaScript Object when the request Content-Type is 'application/json'. 	Version 2015 Release 2

SuiteScript 2.0 Scheduled Script Type

Scheduled scripts are server-side scripts that are executed (processed) with [SuiteCloud Processors](#). You can deploy scheduled scripts so they are submitted for processing at a future time, or at future times

on a recurring basis. You can also submit scheduled scripts on demand from the deployment record or from another script with the [task.ScheduledScriptTask](#) API.

For additional information about SuiteScript 2.0 scheduled scripts, see the following:

- [SuiteScript 2.0 Scheduled Script Reference](#)
 - [SuiteScript 2.0 Scheduled Script Submission](#)
 - [Scheduled Script Deployment Record](#)
 - [Scheduled Script Deployments that Continue to Use Queues](#)
 - [Scheduling a One Time or Recurring Scheduled Script Submission](#)
 - [Submitting an On Demand Scheduled Script Instance from the UI](#)
 - [Submitting an On Demand Scheduled Script Instance from a Script](#)
 - [SuiteScript 2.0 Scheduled Script Execution](#)
 - [SuiteScript 2.0 Scheduled Script Debugging](#)
 - [SuiteScript 2.0 Scheduled Script Status Page](#)
 - [SuiteScript 2.0 Scheduled Script Handling of Server Restarts](#)
- [SuiteScript 2.0 Script Entry Points and API](#)
 - [execute](#)
 - [context.InvocationType](#)

Scheduled Script Use Cases

Use this script type for basis scheduled or on demand tasks. Your SuiteScript 2.0 scheduled script should not process a large amount of data or a large number of records. It should not be used for operations that are long running.

For example, use this script type if:

- You need to log basic information on a recurring schedule
- You need to schedule the future execution of a maintenance script
- You need to create and then purge temporary records
- You need to asynchronously execute logic within another server-side script

Note: If you previously used SuiteScript 1.0 scheduled scripts, many of the use cases for those scripts now apply to the SuiteScript 2.0 [SuiteScript 2.0 Map/Reduce Script Type](#).

Scheduled Script Governance

Each scheduled script instance can use a maximum of 10,000 usage units. For additional information on governance and usage units, see the help topic [SuiteScript Governance](#).

With SuiteScript 2.0 scheduled scripts, you cannot set recovery points and you do not have the ability to yield. There is no SuiteScript 2.0 equivalent to the SuiteScript 1.0 [nlapiYieldScript\(\)](#) and [nlapiSetRecoveryPoint\(\)](#) APIs. If you need to process a large amount of data or a large number of records, use the [SuiteScript 2.0 Map/Reduce Script Type](#) instead. The map/reduce script type has built in yielding and can be submitted for processing in the same ways as scheduled scripts.

Scheduled Script Entry Points

Script Entry Point	
<code>execute</code>	Definition of the scheduled script trigger point

Scheduled Script API

API	
<code>context.InvocationType</code>	Enumeration that holds the string values for scheduled script execution contexts.

Scheduled Script Sample

This sample script finds and fulfills sales orders created on the current day.

Before you submit this script:

1. Create a sales order type of saved search. You can use `search.create(options)` and the `search.Type` enum to set up a saved search with the correct search id and search type.
2. Create a script parameter on the script record **Parameters** subtab. The sample accepts a search id from a script parameter that it assumes was created with the script record.
 - Set the id to `custscript_searchid`.
 - Set Type to Free-Form Text.
 - Assign the saved search id to the parameter. This is done on the deployment record **Parameters** subtab,

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType ScheduledScript
 */
define(['N/search', 'N/record', 'N/email', 'N/runtime'],
function(search, record, email, runtime) {
  function execute(context) {
    if (context.type !== context.InvocationType.ON_DEMAND)
      return;
    var searchId = runtime.getCurrentScript().getParameter("custscript_searchid");
    try {
      search.load({
        id: searchId
      }).run().each(function(result) {
        log.debug({
          details: 'transforming so : ' + result.id + ' to item fulfillment'
        });
        var fulfillmentRecord = record.transform({
          fromType: record.Type.SALES_ORDER,
          fromId: result.id,
          toType: record.Type.ITEM_FULFILLMENT,
          isDynamic: false
        });
      });
    } catch (err) {
      log.error(err);
    }
  }
});
```

```

    });
    var lineCount = fulfillmentRecord.getLineCount('item');
    for (var i = 0; i < lineCount; i++) {
        fulfillmentRecord.setSublistValue('item', 'location', i, 1);
    }
    var fulfillmentId = fulfillmentRecord.save();
    var so = record.load({
        type: record.Type.SALES_ORDER,
        id: result.id
    });
    so.setValue('memo', fulfillmentId);
    so.save();
    return true;
});
} catch (e) {
    var subject = 'Fatal Error: Unable to transform salesorder to item fulfillment!';
    var authorId = -5;
    var recipientEmail = 'notify@company.com';
    email.send({
        author: authorId,
        recipients: recipientEmail,
        subject: subject,
        body: 'Fatal error occurred in script: ' + runtime.getCurrentScript().id + '\n\n' + JSON.stringify(e)
    });
}
}
return {
    execute: execute
};
});

```

SuiteScript 2.0 Scheduled Script Reference

- [SuiteScript 2.0 Scheduled Script Submission](#)
- [SuiteScript 2.0 Scheduled Script Execution](#)
- [SuiteScript 2.0 Scheduled Script Debugging](#)
- [SuiteScript 2.0 Scheduled Script Status Page](#)
- [SuiteScript 2.0 Scheduled Script Handling of Server Restarts](#)

SuiteScript 2.0 Scheduled Script Submission

All Scheduled script instances are executed (processed) by [SuiteCloud Processors](#). You can submit a scheduled script instance for processing in one of three ways:

- By [Scheduling a One Time or Recurring Scheduled Script Submission](#) from the script deployment record UI
- By [Submitting an On Demand Scheduled Script Instance from the UI](#) with the **Save and Execute** option
- By [Submitting an On Demand Scheduled Script Instance from a Script](#) with the `task.ScheduledScriptTask` API

Each of these options requires you to create a [Scheduled Script Deployment Record](#).



Important: After a scheduled script instance is submitted for processing, there may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may need to wait until other scripts have completed.

Scheduled Script Deployment Record

Before you can submit a scheduled script instance to [SuiteCloud Processors](#), you must first create at least one deployment record for the script.

Body Fields

Field	Description
Script	<p>A link to the script record associated with the deployment</p> <div>  Note: This value cannot be changed, even on a new deployment record. If you begin the process of creating a deployment and realize that you have selected the wrong script record, you must start the process over. </div>
Title	The user-defined name for the deployment
ID	<p>A unique identifier for the deployment:</p> <ul style="list-style-type: none"> ■ If this field is left blank when the deployment is created, the system generates the ID. ■ You should enter a custom ID if you plan to bundle the script and deployment for installation into another account. This reduces the risk of naming conflicts. Custom IDs must be all lower case and cannot contain spaces. You can use “_” as a delineator. <div>  Note: For both custom and system-defined IDs, the system appends the string “customdeploy” to the ID. </div> <p>Although not recommended, you can edit this value on an existing deployment. To do this, click Edit on the deployment record. Then click Change ID.</p>
Deployed	<p>Indicates whether the deployment is active. This box must be enabled if you want your script to execute.</p> <p>If you disable this option:</p> <ul style="list-style-type: none"> ■ The script does not execute, regardless of the status. This applies even if a submission schedule is configured on the Schedule subtab. ■ The Save and Execute option no longer displays on the Save dropdown when the deployment record is in edit mode. ■ The script cannot be submitted programmatically. For more information, see Submitting an On Demand Scheduled Script Instance from a Script.
Status	<p>This value determines how and when a script can be submitted to SuiteCloud Processors for processing. Possible options are:</p> <ul style="list-style-type: none"> ■ Testing: You can test the script in the SuiteScript Debugger by using Deployed Debugging. Deployed debugging is restricted to the script owner. ■ Scheduled: You can schedule a single or recurring automatic submission of the script on the Schedule subtab. You cannot submit an on demand instance of the script when it has this status.

Field	Description
	<div>  Note: If you schedule a recurring submission of the script, the script's deployment status on the SuiteScript 2.0 Scheduled Script Status Page remains as Scheduled, even after the script completes its execution. </div> <ul style="list-style-type: none"> ■ Not Scheduled: The script is not currently scheduled for automatic submission. You can submit an on demand instance of the script with either: <ul style="list-style-type: none"> □ The Save and Execute button on the deployment record □ The <code>task.ScheduledScriptTask</code> API You can submit an on demand instance of the script only if there is no other unfinished instance of the same script.
See Instances	A link to the SuiteScript 2.0 Scheduled Script Status Page , filtered for all deployment instances of the script associated with this particular deployment.
Log Level	<p>This value determines the information logged for this deployment. Entries are displayed on the Execution Log subtab. Possible options are:</p> <ul style="list-style-type: none"> ■ Debug: Generally set when a script is in testing mode. A log level set to Debug shows all Audit, Error, and Emergency information in the script log. ■ Audit: Generally set for scripts running in production mode. A log level set to Audit provides a record of events that have occurred during the processing of the script (for example, "A request was made to an external site"). ■ Error: Generally used for scripts running in production mode. A log level set to Error shows only unexpected script errors in the script log. ■ Emergency: Generally used for scripts running in production mode. A log level set to Emergency shows only the most critical errors in the script log.
Execute as Role	For scheduled script deployments, this value is automatically set to Administrator and cannot be edited.
Priority	<p>This value impacts when the SuiteCloud Processors scheduler sends the scheduled script job to the processor pool. By default, this field is set to Standard. For additional information, see the help topic SuiteCloud Processors – Priority Levels</p> <div>  Note: Each SuiteScript 2.0 scheduled script instance is handled by a single job within SuiteCloud Processors. </div> <div>  Important: You must understand SuiteCloud Processors before you change this setting. </div> <p>Possible options are:</p> <ul style="list-style-type: none"> ■ High: Use to mark critical jobs that require more immediate processing. The scheduler sends these jobs to the processor pool first. ■ Standard: This is the default setting. It is considered to be a medium priority level. The scheduler sends these jobs to the processor pool if there are no high priority jobs waiting. ■ Low: Use to mark jobs that can tolerate a longer wait time. The scheduler sends these jobs to the processor pool if there are no high or standard priority jobs waiting.
Queue (Deprecated)	This field is deprecated with the introduction of SuiteCloud Processors . After you click Remove Queue , it no longer displays on the deployment record.

Schedule Subtab

Field	Description
Single Event	The scheduled script task is submitted only once. Use this option if you want to schedule a future one time submission.
Daily Event	The scheduled script task is submitted every x number of days. If you schedule the submission to recur every x minutes or hours, the schedule will start over on the next scheduled day. For example, your deployment is set to submit daily, starting at 3:00 am and recurring every five hours. A scheduled script instance is submitted at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. At midnight, the schedule starts over and the next submission is at 3:00 am.
Weekly Event	The scheduled script instance is submitted at least once per scheduled week. If you schedule the submission to recur every x minutes or hours, the schedule will start over on the next scheduled day. For example, your deployment is set to submit on Tuesday and Wednesday, starting at 3:00 am and recurring every five hours. A scheduled script instance is submitted on Tuesday at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. On Wednesday, the schedule starts over and the next submission is at 3:00 am.
Monthly Event	The scheduled script instance is submitted at least once per scheduled month.
Yearly Event	The scheduled script instance is submitted at least once per year.
Start Date	The first submission occurs on this date. This field is mandatory if a one time or recurring schedule is set.
Start Time	If a value is selected, the first submission occurs at this time.
Repeat	If a value is selected, the first submission occurs on the date and time selected. A new script instance is then created and submitted every x minutes or hours until the end of the start date. If applicable, the schedule starts over on the next scheduled day. For example, your deployment is set to submit on Tuesday and Wednesday, starting at 3:00 am and recurring every five hours. A scheduled script instance is submitted on Tuesday at 3:00 am, 8:00 am, 1:00 pm, 6:00 pm, and 11:00 pm. On Wednesday, the schedule starts over and the next submission is at 3:00 am.
End By	If a value is entered, the last submission occurs on this date. If you schedule the submission to recur every x minutes or hours, a new script instance is created and submitted every x minutes or hours until the end of the end date .
No End Date	The schedule does not have a set end date.

Scheduled Script Deployments that Continue to Use Queues

For all scheduled script deployments created **prior** to the introduction of [SuiteCloud Processors](#) , the **Queue** field remains by default. This applies to accounts with and without SuiteCloud Plus. You have control over whether to stop using queues. The deployment record includes a **Remove Queue** option. After you select this option, the deployment no longer uses a queue **and cannot revert back to using a queue**.

The **Queue** field remains to accommodate deployments that rely on the FIFO order of processing imposed by an individual queue. However, all jobs that use queues are actually processed by the same processor pool that handles the jobs that do not use queues. All jobs compete with each other using the same common processing algorithm.

Note: For deployments that continue to use queues, all jobs assigned to the same queue should have the same priority. In most cases, you can keep the default (standard) priority of these jobs. However, in some cases, you may want to change these jobs to a higher or lower priority. One scenario is if you want to ensure that a specific queue always has a processor available. In that case, designate the jobs assigned to that queue as high priority. Alternatively, if you have a group of lower priority jobs, you can designate them as low priority and assign them to the same queue. That will ensure that only one is processed at a time.

Important: If your existing scheduled script deployments rely on implicit dependencies imposed by queues, you should update and test these scripts before you remove queues. Your scripts may be impacted if they rely on the sequence of FIFO (first in, first out).

One possible solution is to programmatically submit scripts in a certain order. To do this, use `task.ScheduledScriptTask` within the first script to submit the second script. This will ensure that the jobs are submitted to the processor pool in the correct order.

Scheduling a One Time or Recurring Scheduled Script Submission

Scheduled script instances can be submitted for processing once at a pre-defined time in the future, or repeatedly on a regular daily, weekly, monthly, or yearly basis. To set a submission scheduled from the deployment record, the Status field must be set to **Scheduled**

Deployment times can be scheduled with a frequency of every 15 minutes, for example 2:00 pm, 2:15 pm, 2:30 pm, and so on.

When you use the Script Deployment page to create the deployment of a script, the times you set on the Schedule subtab are the times the script is being submitted for processing. **The times you set on the Schedule subtab are not necessarily the times the script will execute.** Script deployment does not mean the script will actually execute precisely at 2:00 pm, 2:15 pm, 2:30 pm, and so on.

A scheduled script's deployment status should be set to **Scheduled** for the following reasons:

- The script was set to Testing, but is now ready for production.
- The script does not need to be executed immediately.
- The script must run at recurring times.

Important: Before you submit a scheduled script instance for processing, you must understand how [SuiteCloud Processors](#) works.

Important: After a scheduled script instance is submitted for processing, there may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may need to wait until other scripts have completed.

To schedule a one time or recurring scheduled script submission:

1. First, create your scheduled script entry point script. This includes your JavaScript file and its associated script record. To review a scheduled script example, see [Scheduled Script Sample](#). If this is your first script, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. On the associated script record, select the **Deploy Script** button.
3. When the script deployment record loads, click the **Deployed** check box if it is not already checked.

4. Review the scheduled script deployment field descriptions at [Scheduled Script Deployment Record](#).
5. Set the Status field to **Scheduled**.
6. Set the remaining body fields.
7. On the [Schedule Subtab](#), set all deployment options.

If you want the schedule to **submit hourly on a 24 hour basis**, use the following sample values as a guide:

- Deployed = checked
- Daily Event = [radio button enabled]
- Repeat every 1 day
- Start Date = [today's date]
- **Start Time = 12:00 am**
- Repeat = every hour
- End By = [blank]
- No End Date = checked
- Status = Scheduled
- Log Level = Error
- Execute as Role = Set to **Administrator**

If the **Start Time** is set to any other time than 12:00 am (for example, it is set to 2:00 pm), the script will start at 2:00 pm, but then finish its hourly execution at 12:00 am. It will not resume until the next day at 2:00 pm.

8. Click **Save**.

Submitting an On Demand Scheduled Script Instance from the UI

Scheduled scripts can be submitted for processing on an on demand basis. To do this, use the **Save and Execute** command on the Script Deployment page. The Status field on the Script Deployment page must be set to either **Not Scheduled** or **Testing**.

The **Testing** status is primarily used to debug and test a script. If you wish to step through the script with the [SuiteScript Debugger](#), you must use [Deployed Debugging](#). You cannot step through an entry point script (or any other script that uses the define statement) with [On Demand Debugging](#).



Important: Before you submit a scheduled script instance for processing, you must understand how [SuiteCloud Processors](#) works.



Important: Even if you initiate an on demand deployment of a script that immediately submits the script for processing, this does not mean the script will execute right away. After a script is submitted for processing, there may be a short system delay before the script is actually executed, even if no scripts are before it. If there are scripts already waiting to be executed, the script may wait to be executed until other scripts have completed.

To submit an on demand scheduled script instance from the UI:

1. First, create your scheduled script entry point script. This includes your JavaScript file and its associated script record. To review a scheduled script example, see [Scheduled Script Sample](#). If this is your first script, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. On the associated script record, select the **Deploy Script** button.

3. When the script deployment record loads, click the **Deployed** check box if it is not already checked.
4. Review the scheduled script deployment field descriptions at [Scheduled Script Deployment Record](#).
5. Set the Status field to **Not Scheduled** or **Testing**.
6. Set the remaining body fields.
7. On the [Schedule Subtab](#), set all deployment options.
8. Click **Save and Execute** from the **Save** dropdown.

Submitting an On Demand Scheduled Script Instance from a Script

You can programmatically submit a scheduled script instance to [SuiteCloud Processors](#) using the `task.ScheduledScriptTask` API.

To schedule a one time or recurring scheduled script submission:

1. First, create your scheduled script entry point script. This includes your JavaScript file and its associated script record. To review a scheduled script example, see [Scheduled Script Sample](#). If this is your first script, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).
2. On the associated script record, select the **Deploy Script** button.
3. When the script deployment record loads, click the **Deployed** check box if it is not already checked.
4. Review the scheduled script deployment field descriptions at [Scheduled Script Deployment Record](#).
5. Set the Status field to **Not Scheduled**.
6. Set the remaining body fields.
7. Click **Save**.
8. In the server-side script where you want to submit the scheduled script instance, call `task.create(options)` to return a `task.ScheduledScriptTask` object:

```
var scriptTask = task.create({taskType: task.TaskType.SCHEDULED_SCRIPT});
```

9. Set the `ScheduledScriptTask.scriptId` and `ScheduledScriptTask.deploymentId` properties:

```
scriptTask.scriptId = 1234;  
scriptTask.deploymentId = 'customdeploy1';
```

10. Call `ScheduledScriptTask.submit()` to submit the scheduled script instance to [SuiteCloud Processors](#).

SuiteScript 2.0 Scheduled Script Execution

All scheduled script instances are executed (processed) by [SuiteCloud Processors](#). For additional information on how scheduled scripts are submitted, see [SuiteScript 2.0 Scheduled Script Submission](#). Each submitted scheduled script instance is handled by one job. A scheduler sends the jobs to a processor pool in a particular order. This order is determined by the [SuiteCloud Processors – Priority Levels](#) and the order of submission. Jobs with a higher priority are sent before jobs with a lower priority. Jobs with the same priority go to the processor pool in the order of submission.

SuiteCloud Processors includes advanced settings that can also impact the order in which jobs are sent to the processor pool. For additional information, see the help topic [SuiteCloud Processors – Priority Elevation and Processor Reservation \(Advanced Settings\)](#).

SuiteScript 2.0 Scheduled Script Debugging

The deployment record **Testing** status is primarily used to debug and test a script. If you wish to step through the script with the [SuiteScript Debugger](#), you must use [Deployed Debugging](#). You cannot step through an entry point script (or any other script that uses the `define` statement) with [On Demand Debugging](#).

SuiteScript 2.0 Scheduled Script Status Page

The Scheduled Script Status page shows the current and past runtime statuses of all scheduled script instances in your account that have submitted in the last 30 days. There are different ways you can access the Scheduled Script Status page. How you access the page determines the view that the page opens with.

Pending	The script is submitted and is waiting to be processed.
Deferred	The script is eligible for processing but has not been processed due to constraints. For example, deferred status occurs when one job must wait for another to finish.
Processing	The script is running.
Retry	The script entered the processing state and failed to complete normally. It is eligible to be retried. There is no need to create a new deployment as the script will be retried automatically. To help determine the failure reason (for example, timeout problem), see the script log.
Complete	The script completed normally.
Cancelled	Due to a NetSuite server error, the script was canceled during or before script execution.
Failed	The script began processing but failed to complete normally. Examine your script for possible errors.

SuiteScript 2.0 Scheduled Script Handling of Server Restarts

Occasionally, a scheduled script failure may occur due to an application server restart. This could be due to a NetSuite update or maintenance, or an unexpected failure of the execution environment. Restarts can terminate an application forcefully at any moment. Therefore, robust scripts need to account for restarts and be able to recover from an unexpected interruption.

In SuiteScript 2.0, in the event of an unexpected system failure, the script is restarted from the beginning. The NetSuite system can automatically detect when a scheduled script is forcefully terminated, and restart the script as soon as the resources required to run the script are available. Note that in SuiteScript 2.0, yields and recovery points are not manually scripted. Therefore, handling a restart situation is simpler.

The following sample scripts demonstrate how restarts impact scheduled script execution:

- [Simple Scheduled Script](#)
- [Example: A Problematic Scheduled Script](#)
- [Example: A Robust Scheduled Script](#)

Simple Scheduled Script

If there is a forceful termination in any part of the script, the script is always restarted from the beginning.

The script can detect a restarted execution by examining the `type` attribute of the context argument. The script is being restarted if the value of this argument is `(context.type === "aborted")`. For more information about the context argument, see [execute context.InvocationType](#).

```
/**
 * @NApiVersion 2.0
 * @NScriptType scheduledscript
 */
define([], function(){
  return {
    execute: function (context)
    {
      if (context.type === 'aborted')
      {
        // I might do something differently
      }
      .
      .
      .
    }
  }
});
```

Example: A Problematic Scheduled Script

A very common pattern seen in scheduled scripts is to perform a search and then processing the results. Consider the following script:

The purpose of this example script is to update each sales order in the system. The `filter1` filter ensures that the search returns exactly one entry per sales order. However, if the script is forcefully interrupted during its processing and then restarted, some sales orders might be updated twice.

To prevent data issues that result from re-processing, the script should use idempotent operations. This means that any operation handled by the script can repeat itself with the same result (e.g. prevent creating duplicate records). Or, the script must be able to recover (e.g. by creating a new task to remove duplicates).

This script does not use idempotent operations, and a large number of sales orders could be updated twice (for example, doubling a price on a sales order from a repeated operation). To improve a script like this, see [Example: A Robust Scheduled Script](#) and [Example 5: A Robust Map/Reduce Script](#).

```
/**
 * @NApiVersion 2.0
 * @NScriptType scheduledscript
 */
define(['N/search', 'N/record'],
  function(search, record){
    return {
      execute: function (context)
      {
        var filter1 = search.createFilter({
          name: 'mainline',
          operator: search.Operator.IS,
          values: true
        });
        var srch = search.create({
```

```

        type: search.Type.SALES_ORDER,
        filters: [filter1],
        columns: []
    });

    var pagedResults = srch.runPaged();

    pagedResults.pageRanges.forEach(function(pageRange){
        var currentPage = pagedResults.fetch({index: pageRange.index});
        currentPage.data.forEach(function(result){
            var so = record.load({
                type: record.Type.SALES_ORDER,
                id: result.id
            });
            //UPDATE so FIELDS>
            so.save()
        });
    });
}
});

```

Example: A Robust Scheduled Script

The following sample code uses `custbody_processed_flag` on the sales order. It is a custom boolean field that must be previously created in the UI. When the sales order is processed, the field is set to `true`. The search then contains an additional filter that excludes flagged sales orders. When the script is restarted, only the sales order which have not been updated are processed.

```

/**
 * @NApiVersion 2.0
 * @NScriptType scheduledscript
 */
define(['N/search', 'N/record'],
    function(search, record){
        return {
            execute: function (context)
            {
                var filter1 = search.createFilter({
                    name: 'mainline',
                    operator: search.Operator.IS,
                    values: true
                });
                var filter2 = search.createFilter({
                    name: 'custbody_processed_flag',
                    operator: search.Operator.IS,
                    values: false
                });
                var srch = search.create({
                    type: search.Type.SALES_ORDER,
                    filters: [filter1, filter2],
                    columns: []
                });

                var pagedResults = srch.runPaged();
            }
        };
    }
);

```

```

        pagedResults.pageRanges.forEach(function(pageRange){
            var currentPage = pagedResults.fetch({index: pageRange.index});
            currentPage.data.forEach(function(result){
                var so = record.load({
                    type: record.Type.SALES_ORDER,
                    id: result.id
                });
                // UPDATE so FIELDS
                so.setValue({
                    fieldID: 'custbody_processed_flag',
                    value: true
                });
                so.save()
            });
        });
    }
});

```

SuiteScript 2.0 Script Entry Points and API

Entry Points:

- `execute`

API:

- `context.InvocationType`

execute

Description	Definition of the scheduled script trigger point
Returns	void
Since	Version 2015 Release 2

Parameters

 **Note:** The `scriptContext` parameter is a JavaScript object. NetSuite automatically passes this object to the script entry point.

Parameter	Type	Required / Optional	Description	Since
<code>scriptContext.type</code>	string	required	The context in which the script is executed. Values are reflected in the <code>context.InvocationType</code> enum.	Version 2015 Release 2

context.InvocationType

Enum Description	Enumeration that holds the string values for scheduled script execution contexts.
-------------------------	---

Module	SuiteScript 2.0 Scheduled Script Type
Since	Version 2015 Release 2

Values

SCHEDULED	The normal execution according to the deployment options specified in the UI.
ON_DEMAND	<p>The script is executed via a call from a script (using ScheduledScriptTask.submit()).</p> <div>  Note: The scheduled script must have a status of Not Scheduled on the Script Deployment page. </div>
USER_INTERFACE	The script is executed via the UI (the Save & Execute button has been clicked).
ABORTED	The script re-executed automatically following an aborted execution (system went down during execution).
SKIPPED	The script is executed automatically following downtime during which the script should have been executed.

SuiteScript 2.0 Suitelet Script Type

Suitelets are extensions of the SuiteScript API that allow you to build custom NetSuite pages and backend logic. Suitelets are server-side scripts that operate in a request-response model, and are invoked by HTTP GET or POST requests to system generated URLs.

There are two types of Suitelets:

1. **Suitelets** use UI objects to create custom pages that look like NetSuite pages. SuiteScript UI objects encapsulate the elements for building NetSuite-looking portlets, forms, fields, sublists, tabs, lists, and columns.
2. **Backend Suitelets** do not use any UI objects and execute backend logic, which can then be parsed by other parts of a custom application. Backend Suitelets are best used for the following purposes:
 - Providing a service for backend logic to other SuiteScripts, or to other external hosts outside of NetSuite.
 - Offloading server logic from client scripts to a backend Suitelet shipped without source code to protect sensitive intellectual property.



Important: RESTlets can be used as an alternative to backend Suitelets.



Note: Suitelets are not intended to work inside web stores. Use online forms to embed forms inside a web store.



Important: The governance limit for concurrent requests for Suitelets available without login is the same as the limit for RESTlets. For information about the account limits, see the help topic [Web Services and RESTlet Concurrency Governance](#). For Suitelets that are authenticated through login, the number of concurrent requests is currently not governed.

See the following for more information about the Suitelet script type:

- [SuiteScript 2.0 Suitelet Script Reference](#)
 - [SuiteScript 2.0 — How Suitelet Scripts are Executed](#)
 - [SuiteScript 2.0 — Reserved Parameter Names in Suitelet URLs](#)
 - [SuiteScript 2.0 Suitelet Script Best Practices](#)
 - [SuiteScript 2.0 Suitelet Script Deployment Page](#)
 - [Embedding HTML in Suitelets](#)
- [SuiteScript 2.0 Suitelet Script Entry Points and API](#)
 - [onRequest\(params\)](#)

SuiteScript 2.0 Suitelet Script Type Sample

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following sample shows how to create a Suitelet form that lets you write and send an email:

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget', 'N/email', 'N/runtime'],
function(ui, email, runtime) {
    function onRequest(context) {
        if (context.request.method === 'GET') {
            var form = ui.createForm({
                title: 'Demo Suitelet Form'
            });
            var subject = form.addField({
                id: 'subject',
                type: ui.FieldType.TEXT,
                label: 'Subject'
            });
            subject.layoutType = ui.FieldLayoutType.NORMAL;
            subject.breakType = ui.FieldBreakType.STARTCOL;
            subject.isMandatory = true;
            var recipient = form.addField({
                id: 'recipient',
                type: ui.FieldType.EMAIL,
                label: 'Recipient email'
            });
            recipient.isMandatory = true;
            var message = form.addField({
                id: 'message',
                type: ui.FieldType.TEXTAREA,
                label: 'Message'
            });
            message.displaySize = {
                width: 60,
                height: 10
            };
            form.addSubmitButton({
                label: 'Send Email'
            });
        }
    }
});
```

```

        context.response.writePage(form);
    } else {
        var request = context.request;
        email.send({
            author: runtime.getCurrentUser().id,
            recipients: request.parameters.recipient,
            subject: request.parameters.subject,
            body: request.parameters.message
        });
    }
}
return {
    onRequest: onRequest
};
});

```

SuiteScript 2.0 Suitelet Script Reference

- [SuiteScript 2.0 — How Suitelet Scripts are Executed](#)
- [SuiteScript 2.0 — Reserved Parameter Names in Suitelet URLs](#)
- [SuiteScript 2.0 Suitelet Script Best Practices](#)
- [SuiteScript 2.0 Suitelet Script Deployment Page](#)
- [Embedding HTML in Suitelets](#)

SuiteScript 2.0 — How Suitelet Scripts are Executed

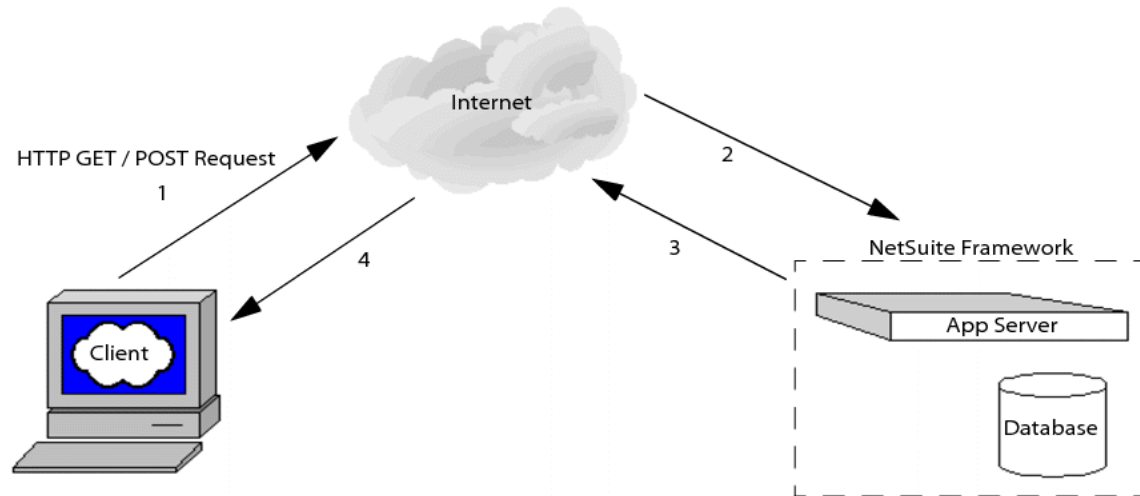
The following steps and diagram provide an overview of the Suitelet execution process:

1. Client initiates an HTTP GET or POST request (typically from a browser) for a system-generated URL. A web request object contains the data from the client's request.
2. The user's script is invoked, which gives the user access to the entire Server SuiteScript API as well as a web request and web response object.
3. NetSuite processes the user's script and returns a web response object to the client. The response can be in following forms:
 - Free-form text
 - HTML
 - RSS
 - XML
 - A browser redirect to another page on the Internet



Important: You can only redirect to external URLs from Suitelets that are accessed externally (in other words, the Suitelet has been designated as “Available Without Login” and is accessed from its external URL).

- A browser redirect to an internal NetSuite page. The NetSuite page can be either a standard page or custom page that has been dynamically generated using UI objects.
4. The data renders in the user's browser.



SuiteScript 2.0 — Reserved Parameter Names in Suitelet URLs

Certain names are reserved and should not be referenced when naming custom parameters for Suitelet URLs.

The following table contains a list of reserved parameter names:

Reserved Suitelet URL Parameter Names	
e	print
id	email
cp	q
l	si
popup	st
s	r
d	displayonly
_nodrop	nodisplay
sc	deploy
sticky	script

If any of your parameters are named after any of the reserved parameter names, your Suitelet may throw an error saying, "There are no records of this type." To avoid naming conflicts, NetSuite recommends that all custom URL parameters are prefixed with **custom**. For example, use `custom_id` instead of `id`.

SuiteScript 2.0 Suitelet Script Best Practices

The following list shows best practices for both form-level and record-level client script development:

- Suitelets are ideal for generating NetSuite pages (forms, lists), returning data (XML, text), and redirecting requests.
- Limit the number of UI objects on a page (< 100 rows for sublists, < 100 options for on demand select fields, < 200 rows for lists).
- Experiment with inline HTML fields embedded on form before going the full custom HTML page route.
- Deploy Suitelets as "Available without Login" only if absolutely necessary (no user context, login performance overhead). (See the help topic [Setting Available Without Login.](#))

- Append “ifrmcntnr=T” to the external URL when embedding in iFrame especially if you are using Firefox. (For more about NetSuite and iFrame, see the help topic [Embedding an Online Form in your Web Site Page](#).)
- When building custom UI outside of the standard NetSuite UI (such as building a custom mobile page using Suitelet), use the User Credentials APIs to help users manage their credentials within the custom UI. For more information, see User Credentials APIs.

SuiteScript 2.0 Suitelet Script Deployment Page

Before a suitelet script can be executed, you must create at least one deployment record for the script.

The deployment record for a suitelet script is similar to that of other script types. However, a suitelet script deployment contains some additional fields. This topic describes all of the available fields.

You can access a suitelet script deployment record in the following ways:

- To open an existing deployment record for editing, go to Customization > Scripting > Script Deployments (Administrator). Locate the appropriate record, and click the corresponding **Edit** link.
- To start creating a new deployment record, open the appropriate script record in view mode, and click the **Deploy Script** button. For help creating a script record, see [Script Record Creation](#).

Suitelet Script Deployment Page Body Fields

The following table summarizes the body fields available on the Suitelet script deployment record. Note that some fields are available only when you edit or view an existing deployment record.

Field	Description
Script	A link to the script record associated with the deployment. This value cannot be changed, even on a new deployment record. If you begin the process of creating a deployment and realize that you selected the wrong script record, you must start the process over.
Title	The user-defined name for the deployment.
ID	<p>A unique identifier for the deployment.</p> <p>On a new record, you can customize this identifier by entering a value in the ID field. You should customize the ID if you plan to bundle the deployment for installation into another account, because using custom IDs helps avoid naming conflicts. IDs must be lowercase and cannot use spaces. You can use an underscore as a delineator.</p> <p>If you do not enter a value, the system automatically generates one. In both cases, the system automatically adds the prefix <code>customdeploy</code> to the ID created when the record is saved. Although not recommended, you can change the ID on an existing deployment by clicking the Change ID button.</p>
Deployed	<p>A configuration option that indicates whether the deployment is active. This box must be checked if you want the script to execute. Otherwise, the system uses the following behavior:</p> <ul style="list-style-type: none"> ■ When the Deployed box is cleared, and the status is Not Scheduled, the Save and Execute option is no longer available, and the deployment cannot be submitted programmatically. ■ When the Deployed box is cleared, and the deployment record's status is Scheduled, any times configured on the Schedule subtab are ignored and the script deployment is not submitted.
Status	<p>A value that determines how and when a script deployment can be submitted for processing. The primary options are:</p> <ul style="list-style-type: none"> ■ Scheduled —The script deployment is submitted for processing at the times indicated on the Schedule subtab.

	<ul style="list-style-type: none"> ■ Not Scheduled — The script deployment is submitted for processing only when invoked manually, either through the UI or programmatically. <p>Note also that, regardless of the Status, the system submits the deployment for processing only if the Deployed box is checked.</p>
Event Type	<p>Use the Event Type list to specify a script execution context at the time of script deployment. After an event type is specified, the deployed script executes only on that event, regardless of the event types specified in the actual script file.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;">  Important: Event types specified in the UI take precedence over the types specified in the script file. For example, if the create event type is specified in the script, selecting delete from the Event Type list restricts the script from running on any event other than delete. If the Event Type field is left blank, your script will execute only on the event type(s) specified in the script file. </div>
Log Level	<p>A value that determines what type of log messages are displayed on the Execution Log of both the deployment record and associated script record. The available levels are:</p> <ul style="list-style-type: none"> ■ Debug — suitable for scripts still being tested; this level shows all debug, audit, error and emergency messages. ■ Use Audit, Error, or Emergency — suitable for scripts in production mode. Of these three, Audit is the most inclusive and Emergency the most exclusive.
Execute As Role	Indicates the role used to run the script.
Available Without Login	Indicates if users without an active NetSuite session can access the Suitelet. See Setting Available Without Login .

Setting Available Without Login

When you select Available Without Login and then save the Script Deployment record, an External URL appears on the Script Deployment page (see figure above). Use this URL for Suitelets you want to make available to users who do not have an active NetSuite session.

Only a subset of the SuiteScript API is supported in externally available Suitelets (Suitelets set to Available Without Login on the Script Deployment page). Note that if you want to use all available SuiteScript APIs in a Suitelet, your Suitelet will require a valid NetSuite session. (A valid session means that users have authenticated to NetSuite by providing their email address and password.)

The following are a few uses cases that address when you might want to make a Suitelet externally available:

- Hosting one-off online forms, such as capturing partner conference registrations.
- Inbound partner communication, such as listening for payment notification responses from PayPal or Google checkout; or for generating the unsubscribe from email campaigns page, which requires access to account information but should not require a login or hosted website.
- For Facebook, Google, and Yahoo mashups in which the Suitelet lives in those websites but needs to communicate to NetSuite via POST requests.

 **Important:** Because there are no login requirements for Suitelets that are available without login, be aware that the data contained within the Suitelet will be less secure.

Errors Related to the Available Without Login URL

You will use either the internal URL or the external URL as the launching point for a Suitelet.

Some factors that determine whether a Suitelet will successfully deploy are:

- Dependencies between the type of URL you are referencing (internal or external)
- The Suitelet deployment status (Testing or Released)
- Whether the Select All checkbox has been selected on the Audience subtab of the Script Deployment Page.

The following table summarizes these dependencies:

Suitelet URL Type	Deployment Status	Select All check boxes	Result
internal	Testing	not checked	Suitelet deploys successfully
internal	Testing	checked	Suitelet deploys successfully
internal	Released	not checked	Error message: You do not have privileges to view this page.
internal	Released	checked	Suitelet deploys successfully
external	Testing	not checked	Error message: You are not allowed to navigate directly to this page.
external	Testing	checked	Error message: You are not allowed to navigate directly to this page.
external	Released	checked	Suitelet deploys successfully
external	Released	not checked	Error message: You do not have privileges to view this page.

Suitelet Script Deployment Page Audience Subtab

Use the Suitelet Script Deployment page's Audience subtab to specify the roles that can access your Suitelet.

The following table summarizes the Audience subtab fields.

Field	Description
Roles	Select the roles that can access your Suitelet. To give access to all roles, check the Select All box.
Groups	Select the Groups that can access your Suitelet.
Partners	Select the Partners that can access your Suitelet. To give access to all partners, check the Select All box.
Departments	Select the Departments that can access your Suitelet.
Employees	Select the Employees that can access your Suitelet. To give access to all employees, check the Select All box.

Suitelet Script Deployment Page Links Subtab

Use the Suitelet Script Deployment page's Links subtab to create links in NetSuite Centers to your Suitelet. For example, create a link to a Suitelet from the Support Section in the Classic Center.

The following table summarizes the Links subtab fields.

Field	Description
-------	-------------

Center	The NetSuite Center's from which users can access your Suitelet.
Section	The NetSuite section of your Suitelet. Users access your Suitelet from this section. For example, Support or Documents.
Category	The name of the Suitelet. Users see this label in the section menu, after the category name. For example, Support > Suitelet Category Name > Your Suitelet Name.
Label	The name of the Suitelet. Users see this label in the section menu, after the category name. For example, Support > Suitelet Category Name > Your Suitelet Name.
Translation	If you have the multi-language feature enabled for your NetSuite account, you can add translations for your Suitelet label in the Translation column.
Insert Before	Specify the section category you would like to insert your Suitelet before.

Embedding HTML in Suitelets

The following examples illustrate how to embed HTML in Suitelet code to add custom elements to a form. Embedding HTML in Suitelets is useful for adding components that are not available through the SuiteScript N/ui/serverWidget module.

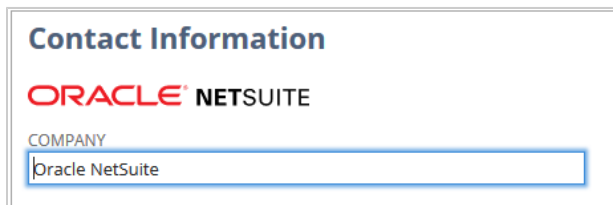
- [Embedding Inline HTML in a Field](#)
- [Embedding HTML from a Linked HTML Page in a Suitelet](#)

Embedding Inline HTML in a Field

In a Suitelet, you can use [Form.addField\(options\)](#) to add inline HTML as a field on a form. Specify the type as `INLINEHTML` and use the [Field.defaultValue](#) property to set the HTML value. For more information, see [N/ui/serverWidget Module](#).

```
var htmlImage = form.addField({
  id: 'custpage_htmlfield',
  type: serverWidget.FieldType.INLINEHTML,
  label: 'HTML Image'
});
htmlImage.defaultValue = "<img src='https://system.netsuite.com/images/logos/netsuite-oracle.svg' alt='Oracle Netsuite logo'>";
```

The following screenshot shows a form with an `INLINEHTML` field containing an image field.



The screenshot displays a web form titled "Contact Information". At the top, it features the "ORACLE NETSUITE" logo. Below the logo is a text input field with the label "COMPANY" above it. The input field contains the text "Oracle NetSuite".

Embedding HTML from a Linked HTML Page in a Suitelet

In a Suitelet, you can use [https.get\(options\)](#) from the N/https module to embed HTML from a linked HTML page in the form created by the Suitelet. The Suitelet manages the data submitted by users to the HTML page. When you use a linked HTML page, you manage HTML code through strings. As a result, it can be harder to manipulate data than it is with components created with N/serverWidget module methods. To pass data through the string containing the HTML code, your Suitelet code must

change values within the string. This string becomes increasingly complex as the number of values increases. For more information, see [N/https Module](#).

The following example creates a simple volunteer sign-up sheet. It consists of two parts: the Suitelet code and the HTML. This Suitelet code uses the [https.get\(options\)](#) method from the N/https module to access the content from the HTML page. It uses methods from the N/record module, N/email module, and N/search module to collect, process, and respond to user-submitted data.

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

define(['N/https', 'N/record', 'N/email', 'N/search'],

function callbackFunction(https, record, email, search) {

    function getFunction(context) {

        var contentRequest = https.get({
            url: "https://LinkToFormPage.html"
        });
        var contentDocument = contentRequest.body;
        var sponsorid = context.request.parameters.sponsorid;

        if (sponsorid && sponsorid != "" && sponsorid != null) {
            contentDocument = contentDocument.replace("{sponsorid}", sponsorid);
            log.debug("Setting Sponsor", sponsorid)
        }

        var projectid = context.request.parameters.projectid;

        if (projectid && projectid != "" && projectid != null) {
            contentDocument = contentDocument.replace("{projectid}", projectid);
            log.debug("Setting Project", projectid);
        }

        context.response.write(contentDocument);
    }

    function postFunction(context) {

        var params = context.request.parameters;

        var emailString = "First Name: {{firstname}}\nLast Name: {{lastname}}\nEmail: {{email}}\nFacebook URL: {{customer_fb_url}}";

        var contactRecord = record.create({
            type: "contact",
            isDynamic: true
        });

        for (param in params) {
            if (param === "company") {
```



```

        if (params[param] !== "{{sponsorid}}") {
            contactRecord.setValue({
                fieldId: param,
                value: params[param]
            });
            var lkpfld = search.lookupFields({
                type: "vendor",
                id: params["company"],
                columns: ["entityid"]
            });
            emailString += "\nSponsor: " + lkpfld.entityid;
        }
        else {
            contactRecord.setValue({
                fieldId: "custentity_sv_shn_isindi",
                value: true
            })
        }
    }
    else {
        if (param !== "project") {
            contactRecord.setValue({
                fieldId: param,
                value: params[param]
            });
            var replacer = "{" + param + "}";
            emailString = emailString.replace(replacer, params[param]);
        }
    }
}

var contactId = contactRecord.save({
    ignoreMandatoryFields: true,
    enableSourcing: true
});

log.debug("Record ID", contactId);

if (params["project"] && params["project"] !== "" && params["project"] != null && params
["project"] !== "{{projectid}}") {

    var lkpfld = search.lookupFields({
        type: "job",
        id: params["project"],
        columns: ["companyname"]
    });

    emailString += "\nProject Name: " + lkpfld.companyname;

    var participationRec = record.create({
        type: "customrecord_project_participants",
        isDynamic: true
    });

    participationRec.setValue({

```

```

        fieldId: "custrecord_participants_volunteer",
        value: contactId
    })

    participationRec.setValue({
        fieldId: "custrecord_participants_project",
        value: params["project"]
    })

    var participationId = participationRec.save({
        enableSourcing: true,
        ignoreMandatoryFields: true
    })
}

log.debug("Email String", emailString);

email.send({
    author: -5,
    recipients: 256,
    subject: "New Volunteer Signed Up",
    body: "A new volunteer has joined:\n\n" + emailString
});

email.send({
    author: -5,
    recipients: params["email"],
    subject: "Thank you!",
    body: "Thank you for volunteering:\n\n" + emailString
});

var contentRequest = https.get({
    url: "https://LinkToFormCompletePage.html"
});

var contentDocument = contentRequest.body;

context.response.write(contentDocument);

}

function onRequestFxn(context) {

    if (context.request.method === "GET") {
        getFunction(context)
    }
    else {
        postFunction(context)
    }
}

return {
    onRequest: onRequestFxn
};
};

```

The following HTML, contained in the LinkToFormPage.html file, creates the form that the Suitelet links to.

```
<form method="post" class="form-horizontal" action="https://LinkToSuitelet.js">
  <table>
    <tbody><tr>
      <td>First Name</td>
      <td class="col-md-8"><input class="form-control" id="firstname" placeholder="First Name" name="firstname"
required="" type="text"></td>
    </tr>
    <tr>
      <td>Last Name</td>
      <td class="col-md-8"><input class="form-control" id="lastname" placeholder="Last Name" name="lastname"
required="" type="text"></td>
    </tr>
    <tr>
      <td>Email</td>
      <td class="col-md-8"><input class="form-control" id="email" placeholder="email" name="email" required=""
type="email"></td>
    </tr>
    <tr>
      <td>Facebook URL</td>
      <td class="col-md-8"><input class="form-control" id="custentity_fb_url" placeholder="Facebook"
name="custentity_fb_url" required="" type="text"></td>
    </tr>
    <tr>
      <td colspan="2">
        <input name="company" value="{{sponsorid}}" type="hidden">
        <input name="project" value="{{projectid}}" type="hidden">
      </td>
    </tr>
  </tbody></table>
  <br>
  <button type="submit" class="btn btn-inverse">Sign Up as a Volunteer</button>
</form>
```

The following screenshot illustrates the form that is created by the linked HTML page.

The screenshot shows a web form with the following elements:

- First Name**: A text input field with the placeholder text "First Name".
- Last Name**: A text input field with the placeholder text "Last Name".
- Email**: A text input field with the placeholder text "email".
- Facebook URL**: A text input field with the placeholder text "Facebook".
- Sign Up as a Volunteer**: A dark blue button with white text, positioned below the input fields.

SuiteScript 2.0 Suitelet Script Entry Points and API

■ onRequest(params)

onRequest(params)

Description	Definition of the Suitelet script trigger point.
-------------	--

Returns	void
Since	Version 2015 Release 2

Parameters

 **Note:** The `params` parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>params.request</code>	<code>http.ServerRequest</code>	required	The incoming request.	Version 2015 Release 2
<code>params.response</code>	<code>http.ServerResponse</code>	required	The Suitelet response.	Version 2015 Release 2

SuiteScript 2.0 User Event Script Type

User event scripts are executed on the NetSuite server. They are executed when users perform certain actions on records, such as create, load, update, copy, delete, or submit. Most standard NetSuite records and custom record types support user event scripts.

User event scripts can be used to perform the following tasks:

- Implement custom validation on records.
- Enforce user-defined data integrity and business rules.
- Perform user-defined permission checking and record restrictions.
- Implement real-time data synchronization.
- Define custom workflows. (redirection and follow-up actions)
- Customize forms.

 **Important:** User event scripts cannot be executed by other user event scripts or by workflows with a **Context** of **User Event Script**. In other words, you cannot chain user event scripts. You can, however, execute a user event script from a call within a scheduled script, a portlet script, or a Suitelet.

For additional information about SuiteScript 2.0 User Event Scripts, see the following:

- [SuiteScript 2.0 User Event Script Reference](#)
 - [How User Events are Executed](#)
 - [User Event Script Best Practices](#)
 - [SuiteScript 2.0 User Event Script Tutorial](#)
- [SuiteScript 2.0 User Event Script Entry Points and API](#)
 - [afterSubmit\(scriptContext\)](#)
 - [beforeLoad\(scriptContext\)](#)
 - [beforeSubmit\(scriptContext\)](#)
 - [context.UserEventType](#)

SuiteScript 2.0 User Event Script Sample

The following sample shows a user event script. This script is designed for use in environments that do not use the Team Selling feature.

When you deploy this script on the customer record, this script creates a follow-up phone call record for every newly created customer record.



Important: Before running this script, you must replace the salesrep internal ID with one specific to your account. Specifically, use an ID that represents an employee who is classified as a sales rep. The sales rep option is located on the Human Resources subtab of the employee record. If you do not replace the ID, the script may not work as expected. Additionally, note that this script is designed to work in environments where the customer record includes a salesrep field. If the Team Selling feature is enabled, the customer record typically will not include a salesrep field.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 */
define(['N/record'],
function(record) {
    function beforeLoad(context) {
        if (context.type !== context.UserEventType.CREATE)
            return;

        var customerRecord = context.newRecord;
        customerRecord.setValue('phone', '555-555-5555');
        if (!customerRecord.getValue('salesrep'))
            customerRecord.setValue('salesrep', 46);
    }

    function beforeSubmit(context) {
        if (context.type !== context.UserEventType.CREATE)
            return;

        var customerRecord = context.newRecord;
        customerRecord.setValue('comments', 'Please follow up with this customer!');
    }

    function afterSubmit(context) {
        if (context.type !== context.UserEventType.CREATE)
            return;

        var customerRecord = context.newRecord;
        if (customerRecord.getValue('salesrep')) {
            var call = record.create({
                type: record.Type.PHONE_CALL,
                isDynamic: true
            });
            call.setValue('title', 'Make follow-up call to new customer');
            call.setValue('assigned', customerRecord.getValue('salesrep'));
            call.setValue('phone', customerRecord.getValue('phone'));
            try {
                var callId = call.save();
                log.debug('Call record created successfully', 'Id: ' + callId);
            } catch (e) {
                log.error(e.name);
            }
        }
    }
});
```

```

    }
  }
}
return {
  beforeLoad: beforeLoad,
  beforeSubmit: beforeSubmit,
  afterSubmit: afterSubmit
};
});

```

SuiteScript 2.0 User Event Script Reference

- [How User Events are Executed](#)
- [User Event Script Best Practices](#)
- [SuiteScript 2.0 User Event Script Tutorial](#)

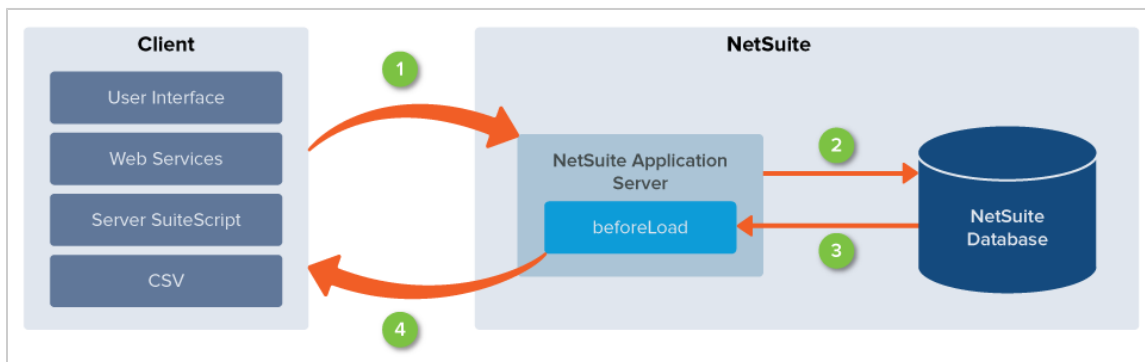
How User Events are Executed

User event scripts are executed based on operation types defined as `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

The following steps and diagram provide an overview of what occurs during a `beforeLoad` operation:

1. The client sends a read operation request for record data. A client request can come from the user interface, web services, CSV import, or server-side SuiteScript (except other user event scripts).
2. Upon receiving the request, the application server performs basic permission checks on the client.
3. The database loads the requested information into the application server for processing. This is where the `beforeLoad` operation occurs – before the requested data is returned to the client.
4. The client receives the now validated/processed `beforeLoad` data.

Note: Standard records cannot be sourced during a `beforeLoad` operation. Use the `pagelnit` client script for this purpose. See [pagelnit](#).



The following steps and diagram provide an overview of what occurs on `beforeSubmit` and `afterSubmit` operations:

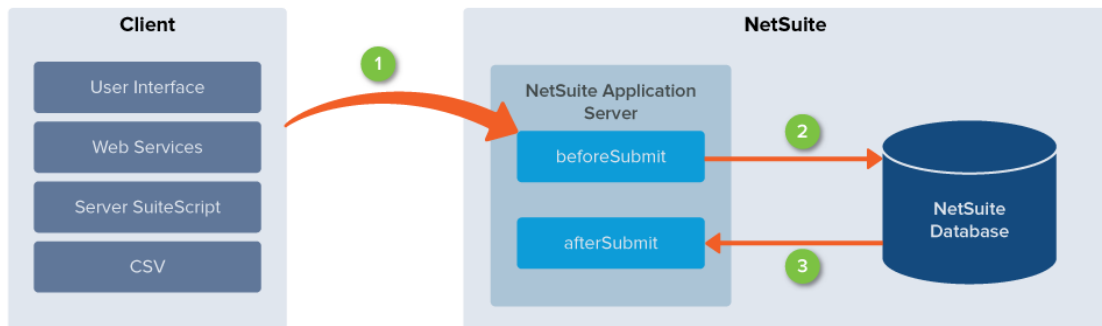
1. The client performs a write operation by submitting data to the application server. (The client request can come from the user interface, web services, server-side SuiteScript calls, CSV imports, or XML.) The application server:

- a. performs basic permission checks on the client
- b. processes the submitted data and performs specified validation checks during a beforeSubmit operation

The submitted data has **NOT** yet been committed to the database.

2. After the data has been validated, it is committed to the database.
3. If this (newly committed) data is then called by an afterSubmit operation, the data is taken from the database and is sent to the application server for additional processing. Examples of afterSubmit operations on data that are already committed to the database include, but are not limited to:
 - a. sending email notifications (regarding the data that was committed to the database)
 - b. creating child records (based on the data that was committed to the database)
 - c. assigning tasks to employees (based on data that was committed to the database)

Note: Asynchronous afterSubmit user events are only supported during webstore checkout.



Tip: You can set the order in which user event scripts execute on the Scripted Records page. See *Using the Scripted Records Page*.

User Event Script Best Practices

The following list shows best practices for user event script development:

- Use the type argument and context object to define and limit the scope of your user event logic.
- Limit the amount of script execution in user event scripts (< 5 seconds) since they run often and in-line. You can use the Script Performance Monitor SuiteApp to test the performance of your scripts deployed on a specific record type.
- Mission critical business logic implemented using user events should be accompanied by a 'Clean up' scheduled script to account for any unexpected errors or mis-fires.
- Any operation that depends on the submitted record being committed to the database should happen in an afterSubmit script.
- Be careful when updating transaction line items in a beforeSubmit script because you have to ensure that the line item totals net taxes and discounts are equal to the summarytotal, discounttotal, shippingtotal, and taxtotal amounts.
- Activities (user events) on a hosted website can trigger server-side SuiteScripts. In addition to Sales Orders, scripts on Case and Customer records also execute because of Web activities.
- Assigning many executable functions to one record type is discouraged because this could negatively affect the user experience with that record type. For example, if there are ten beforeLoad

scripts that must complete their execution before the record loads into the browser, the time needed to load the record may increase significantly. Be aware of the number of user event scripts used, including bundled user event scripts.

- To set a field on a record or make any changes to a record being submitted, use the `beforeSubmit` event rather than the `afterSubmit` event. Changes made during an `afterSubmit` event duplicates the record.
- Perform all post-processing operations of the current record on an `afterSubmit` event.

SuiteScript 2.0 User Event Script Tutorial

A user event script is a server-side script that is triggered by actions taken with a record. This script type can include logic that is executed at any of the following points: after a user has started the process of opening the record but before the record loads, after the user clicks Save, and after the record is submitted to the database.

This topic walks you through the implementation of a basic user event script. It contains the following sections:

- [Sample Script Overview](#)
- [Step One: Check Your Prerequisites](#)
- [Step Two: Create the Script File](#)
- [Step Three: Review the Script \(Optional\)](#)
- [Step Three: Upload the Script File to NetSuite](#)
- [Step Five: Create a Script Record and Script Deployment Record](#)
- [Step Six: Test the Script](#)

 **Note:** Before proceeding, review [SuiteScript 2.0 Script Basics](#) for an explanation of terms used in this topic. For an overview of the basic structural elements required in any SuiteScript 2.0 entry point script, see [SuiteScript 2.0 Anatomy of a Script](#).

Sample Script Overview

The sample script is meant to be deployed on the employee record. When it is, it takes the following actions:

- When a user begins the process of creating a new employee record, the script disables the Notes field before the page loads.
- After the user enters all required information about the employee and clicks Save, the script adds a value to the Notes field. This message states that no date has yet been set for the new employee's orientation.
- After the record has been submitted, the script creates a task record for scheduling the new employee's orientation session. The script assigns the task to the new employee's supervisor.

The script uses all three of the entry points available with the [SuiteScript 2.0 User Event Script Type](#). For details about these entry points, see [beforeLoad\(scriptContext\)](#), [beforeSubmit\(scriptContext\)](#), and [afterSubmit\(scriptContext\)](#).

Step One: Check Your Prerequisites

To complete this tutorial, your system has to be set up properly. Review the following before you proceed:

- [Enable the Feature](#)
- [Create an Employee Record](#)

Enable the Feature

Before you can complete the rest of the steps listed in this topic, the Client and Server SuiteScript features must be enabled in your account. For help enabling these features, see the help topic [Enabling SuiteScript](#).

Create an Employee Record

To complete this tutorial, your system must have an employee record that you can designate as the supervisor of a new test employee.

To view your existing employee records, select Lists > Employees. If your account does not already have an employee record you can use, create one. For help creating an employee record, see the help topic [Adding an Employee](#).

Step Two: Create the Script File

Before proceeding, you must create the script file. To create the file, copy and paste the following code into the text editor of your choice. Save the file and name it createTask.js.

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

// Load two standard modules.

define ( ['N/record', 'N/ui/serverWidget' ] ,

    // Add the callback function.

    function(record, serverWidget) {

        // In the beforeLoad function, disable the Notes field.

        function myBeforeLoad (context) {

            if (context.type !== context.UserEventType.CREATE)
                return;

            var form = context.form;

            var notesField = form.getField({
                id: 'comments'
            });

            notesField.updateDisplayType({
                displayType: serverWidget.FieldDisplayType.DISABLED
            });
        }

        // In the beforeSubmit function, add test to the Notes field.

        function myBeforeSubmit(context) {
```

```

    if (context.type !== context.UserEventType.CREATE)
        return;

    var newEmployeeRecord = context.newRecord;

    newEmployeeRecord.setValue({
        fieldId: 'comments',
        value: 'Orientation date TBD.'
    });
}

// In the afterSubmit function, begin creating a task record.

function myAfterSubmit(context) {

    if (context.type !== context.UserEventType.CREATE)
        return;

    var newEmployeeRecord = context.newRecord;

    var newEmployeeFirstName = newEmployeeRecord.getValue ({
        fieldId: 'firstname'
    });

    var newEmployeeLastName = newEmployeeRecord.getValue ({
        fieldId: 'lastname'
    });

    var newEmployeeSupervisor = newEmployeeRecord.getValue ({
        fieldId: 'supervisor'
    });

    if (newEmployeeSupervisor) {

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

        newTask.setValue({
            fieldId: 'title',
            value: 'Schedule orientation session for ' +
                newEmployeeFirstName + ' ' + newEmployeeLastName
        });

        newTask.setValue({
            fieldId: 'assigned',
            value: newEmployeeSupervisor
        });
    }
}

```

```

    try {

        var newTaskId = newTask.save();

        log.debug({
            title: 'Task record created successfully',
            details: 'New task record ID: ' + newTaskId
        });

    } catch (e) {
        log.error({
            title: e.name,
            details: e.message
        });
    }
}

// Add the return statement that identifies the entry point funtions.

return {
    beforeLoad: myBeforeLoad,
    beforeSubmit: myBeforeSubmit,
    afterSubmit: myAfterSubmit
};
};

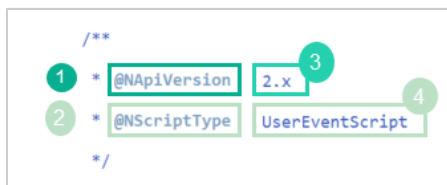
```

Step Three: Review the Script (Optional)

If you want to understand more about how this script is structured, review the following subsections.

JSDoc Tags

The following image shows the JSDoc block used in this sample script. For an explanation of the numbered callouts, see the table that follows the image.



```

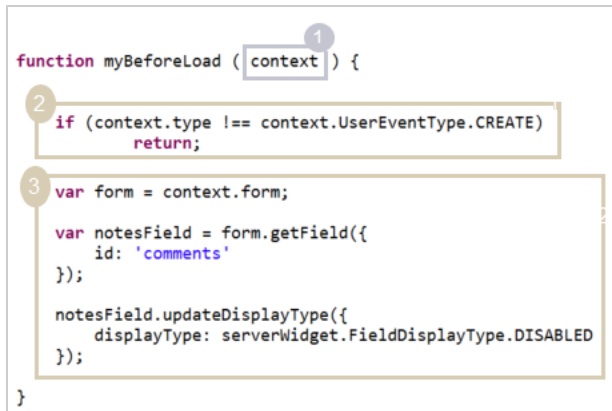
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 */

```

Callout	Description
1 and 3	The @NApiVersion tag and its value (2.x). This tag is required in an entry point scripts. Valid values are 2.0, 2.x, and 2.X.
2 and 4	The @NScriptType tag and its value (UserEventScript). This tag identifies the script type being used. The value is not case sensitive, but using Pascal case, as shown in this example, is recommended for better readability.

beforeLoad Function

The following image shows the function named `myBeforeLoad`. Because of the way the [return Statement](#) is set up, this function executes when the `beforeLoad` entry point is invoked.



```
function myBeforeLoad (context) {
  if (context.type !== context.UserEventType.CREATE)
    return;
  var form = context.form;
  var notesField = form.getField({
    id: 'comments'
  });
  notesField.updateDisplayType({
    displayType: serverWidget.FieldDisplayType.DISABLED
  });
}
```

Callout 1 points to the `context` parameter in the function signature. Callout 2 points to the `if` statement that checks `context.type` and returns if it's not `CREATE`. Callout 3 points to the `form` variable and the subsequent `getField` and `updateDisplayType` calls.

Callout	Description
1	The context object that is made available when the <code>beforeLoad</code> entry point is invoked. You can see a list of the properties that are available to this context object in beforeLoad(scriptContext) . This function uses two of these properties: type and form .
2	This statement tells the system that the rest of the logic in the function should execute only if the user is creating a new record. The statement uses the context object's type property to identify the type of action the user is taking. To see a list of the possible actions, review context.UserEventType .
3	These statements tell the system to disable the Notes field. They do so in part by using the context object's form property, which gives the script access to the form. These statements also use N/ui/serverWidget Module APIs, which let the script access specific fields and change how they appear and behave.

beforeSubmit Function

The following screenshot shows the function named `myBeforeSubmit`. Because of the way the script's [return Statement](#) is structured, this function executes when the `beforeSubmit` entry point is invoked.



```
function myBeforeSubmit (context) {
  if (context.type !== context.UserEventType.CREATE)
    return;
  var newEmployeeRecord = context.newRecord;
  newEmployeeRecord.setValue({
    fieldId: 'comments',
    value: 'Orientation date TBD.'
  });
}
```

Callout 1 points to the `context` parameter in the function signature. Callout 2 points to the `if` statement that checks `context.type` and returns if it's not `CREATE`. Callout 3 points to the `newRecord` variable and the `setValue` call.

Callout	Description
1	The context object that is made available when the <code>beforeSubmit</code> entry point is invoked. You can see a list of the properties that are available to this context object in beforeSubmit(scriptContext) . This function uses two of those properties: type and newRecord .

Callout	Description
	Note that the properties available to this object are different from those available to the beforeLoad Function , which is linked to the beforeLoad entry point. The beforeLoad entry point has access to a form property, but the beforeSubmit object does not. That means that the beforeSubmit entry point function does not have access to the form used by the record.
2	This statement tells the system that the rest of the logic in the function should execute only if the user is creating a new record. To identify the type of action the user is taking, this statement uses the context object's type property.
3	These statements add text to the Notes field of the new record. They do so by using the context object's newRecord property, which gives the script access to the record that is about to be created. This snippet uses the Record.setValue(options) method to add text to the Notes field of the new record.

afterSubmit Function

The following screenshots show the function named myAfterSubmit. Because of the way the script's [return Statement](#) is structured, this function executes when the afterSubmit entry point is invoked.

The following image shows the first part of the function.

```

function myAfterSubmit ( context ) {
    if (context.type !== context.UserEventType.CREATE)
        return;

    var newEmployeeRecord = context.newRecord;
    var newEmployeeFirstName = newEmployeeRecord.getValue ({
        fieldId: 'firstname'
    });
    var newEmployeeLastName = newEmployeeRecord.getValue ({
        fieldId: 'lastname'
    });
    var newEmployeeSupervisor = newEmployeeRecord.getValue ({
        fieldId: 'supervisor'
    });
}

```

The screenshot shows the first part of the `myAfterSubmit` function. Callout 1 points to the `context` parameter in the function signature. Callout 2 points to the `if` statement that checks if the user is creating a new record. Callout 3 points to the block of code that retrieves data from the new record using the `newRecord` property and the `getValue` method.

Callout	Description
1	The context object that is made available when the afterSubmit entry point is invoked. You can see a list of the properties that are available to this context object in afterSubmit(scriptContext) . This function uses two of those properties: type and newRecord .
2	This statement tells the system that the rest of the logic in the function should execute only if the user is creating a new record. To identify the type of action the user is taking, this statement uses the context object's type property.
3	These statements retrieve several pieces of data from the record. They do so by using the context object's newRecord property, which gives the script access to the record that has just been submitted. This snippet also uses the Record.getValue(options) method.

The remaining portion of the function creates the task record.

```

4
if (newEmployeeSupervisor) {
5
    var newTask = record.create({
        type: record.Type.TASK,
        isDynamic: true
    });

    newTask.setValue({
        fieldId: 'title',
        value: 'Schedule orientation session for '
            + newEmployeeFirstName + ' ' + newEmployeeLastName
    });

    newTask.setValue({
        fieldId: 'assigned',
        value: newEmployeeSupervisor
    });

6
    try {
        var newTaskId = newTask.save();

        log.debug({
            title: 'Task record created successfully',
            details: 'New task record ID: ' + newTaskId
        });
    } catch (e) {
        log.error({
            title: e.name,
            details: e.message
        });
    }
}
}

```

Callout	Description
4	A conditional statement. This expression inside the parentheses evaluates to true if a value has been set for the Supervisor field. If this condition is met, the system executes the logic within the curly braces.
5	These statements use N/record Module APIs to create a task record and set values on the record.
6	This code attempts to save the new task record. If any errors are encountered, details are written to the script deployment record's execution log.

return Statement

The following image shows the callback function's return statement.

```

return {
1
    beforeLoad : myBeforeLoad, 2
    beforeSubmit : myBeforeSubmit,
    afterSubmit : myAfterSubmit
};

```

Callout	Description
1	Entry points. The callback function's return statement must use at least one entry point that belongs to the script type identified by the <code>@NScriptType</code> tag (Callout 4 in JSDoc Tags).

Callout	Description
	This script uses all three of the user event script type's entry points.
2	Entry points functions. For each entry point used, your script must identify an entry point function that is defined elsewhere within the script.

Step Four: Upload the Script File to NetSuite

After you have created your entry point script file, upload it to your NetSuite File Cabinet.

To upload the script file:

1. In the NetSuite UI, go to Documents > File > SuiteScripts.
2. Click **Add File**.
3. Follow the prompts to locate the createTask.js file in your local environment and upload it.

Be aware that even after you upload the file, you can edit it from within the File Cabinet, if needed. For details, see the help topic [Editing Files in the File Cabinet](#).

Step Five: Create a Script Record and Script Deployment Record

In general, before an entry point script can execute in your account, you must first create a script record that represents the entry point script file. You must also create a script deployment record.

To create the script record and script deployment record:

1. Go to Customization > Scripting > Scripts > New Customization > Scripting > Scripts > New.
2. In the **Script File** dropdown list, select createTask.js.
Note that, if you had not yet uploaded the file, as described in Step Three, you could upload the file from this page. With your cursor, point to right of the dropdown list to display a plus icon. Clicking this icon opens a window that lets you upload a file.
3. After you have populated the dropdown list, click the **Create Script Record** button.
In response, the system displays a new script record, with the createTask.js file listed on the **Scripts** subtab.
4. Fill out the required body fields as follows:
 - In the **Name** field, enter **Create Task Record User Event**.
 - In the **ID** field, enter **_ues_create_task_record**.
5. Click the **Deployments** subtab.
6. Add a line to the sublist, as follows:
 - Set the **Applies to** dropdown list to **Employee**.
 - In the **ID** field, enter **_ues_create_task_record**.

Leave the other fields set to their default values. Note that the **Status** field is set to **Testing**. This value means that the script does not deploy for other users. (If you wanted to change the deployment later and make the customization's behavior available to all users, you could edit the deployment and set the status to **Released**.)
7. Click **Save**.

The system creates the script and script deployment records.

Step Six: Test the Script

Now that the script is deployed, you should verify that it executes as expected.

To test the script:

1. Begin the process of creating a new employee record by selecting **Lists > Employees > Employees > New**.
2. In the new employee form, check to see whether the **Notes** field is disabled. If the `beforeSubmit` entry point function works as expected, the field is gray and cannot be edited.
3. Enter value for required fields. These fields may vary depending on the features enabled in your account and any customizations that exist. Minimally, you must enter values for the following:
 - **Name** — Enter a first name of John and last name Smith.
 - **Subsidiary** — (OneWorld only) Choose an appropriate value from the dropdown list.
4. To make sure that the `afterSubmit` function is invoked, enter a value in the **Supervisor** field, if you have not already.
5. Click **Save**. A success message appears, and the system displays the new record in View mode.
6. Verify that the `beforeSubmit` function was successful: Look at the **Notes** field. It should include the value **Orientation date TBD**.

Confirmation
Employee successfully Saved

Employee

John Smith **ACTIVE**

[Edit](#) [Back](#) [Actions](#)

Primary Information

EMPLOYEE ID John Smith	JOB TITLE	NOTES Orientation date TBD.
MR./MS...	SUPERVISOR Jane Miller	IMAGE

7. Verify that the `afterSubmit` function was successful:
 - a. Select **Activities > Scheduling > Tasks**.
 - b. Check the filtering options to make sure that your view includes tasks assigned to all users.
 - c. Verify that a task was created and assigned to the person you named as a supervisor in Step 4.

Tasks List Search Audit Trail

VIEW Basic [Customize View](#) [New Task](#)

FILTERS

[EDIT](#) QUICK SORT

EDIT VIEW	INTERNAL ID	DUE DATE	TASK TITLE	COMPANY	PRIORITY	STATUS	ASSIGNED TO
Edit View	509	11/9/2017	Schedule orientation session for John Smith		Medium	Not Started	Jane Miller

Next Steps

If you want to learn how to customize this entry point script so that it calls a custom module, see [SuiteScript 2.0 Custom Module Tutorial](#).

SuiteScript 2.0 User Event Script Entry Points and API



Important: User event scripts cannot be executed by other user event scripts or by workflows with a **Context** of **User Event Script**. In other words, you cannot chain user event scripts. You can, however, execute a user event script from a call within a scheduled script, a portlet script, or a Suitelet.

Script Entry Point	Description
<code>afterSubmit(scriptContext)</code>	Executed immediately after a write operation on a record
<code>beforeLoad(scriptContext)</code>	Executed whenever a read operation occurs on a record, and prior to returning the record or page.
<code>beforeSubmit(scriptContext)</code>	Executed prior to any write operation on the record.
<code>context.UserEventType</code>	Holds the string values for user event execution contexts.

afterSubmit(scriptContext)

Description	<p>Executed immediately after a write operation on a record. The <code>afterSubmit</code> operation is useful for performing any actions that need to occur following a write operation on a record. Examples of these actions include email notification, browser redirect, creation of dependent records, and synchronization with an external system.</p> <p>Notes:</p> <ul style="list-style-type: none"> ■ The approve, cancel, and reject argument types are only available for record types such as sales orders, expense reports, timebills, purchase orders, and return authorizations. ■ Attaching a child custom record to its parent or detaching a child custom record from its parent triggers an edit event. ■ Asynchronous <code>afterSubmit</code> user events are only supported during webstore checkout. This event can be used with the following <code>context.UserEventType</code> s: <ul style="list-style-type: none"> ■ create ■ edit ■ xedit (inline editing; only returns the fields edited and not the full record) ■ delete ■ approve (only available for certain record types) ■ cancel (only available for certain record types) ■ reject (only available for certain record types) ■ pack (only available for certain record types, for example Item Fulfillment records) ■ ship (only available for certain record types, for example Item Fulfillment records) ■ dropship (for purchase orders with items specified as "drop ship") ■ specialorder (for purchase orders with items specified as "special order") ■ orderitems (for purchase orders with items specified as "order item") ■ paybills (use this type to trigger <code>afterSubmit</code> user events for Vendor Payments from the Pay Bill page. Note that no sublist line item information will be available. Users must do a lookup/search to access line item values.)
Returns	void
Since	Version 2015 Release 2

Parameters

Note: The `scriptContext` parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>scriptContext.newRecord</code>	record.Record	required	The new record in read only mode. To edit a record, use the record.load(options) method to load the newly submitted record. Make changes, and submit the record again.	Version 2015 Release 2
<code>scriptContext.oldRecord</code>	record.Record	required	The old record in read only mode.	Version 2015 Release 2
<code>scriptContext.type</code>	string	required	The trigger type. Use the context.UserEventType enum to set this value.	Version 2015 Release 2

beforeLoad(scriptContext)

Description	<p>Executed whenever a read operation occurs on a record, and prior to returning the record or page. These operations include navigating to a record in the UI, reading a record in web services, and loading a record. This event cannot be used to source standard records. Use the <code>pageInit</code> client script for this purpose.</p> <p>Notes:</p> <ul style="list-style-type: none"> beforeLoad user events cannot be triggered when you load/access an online form. Data cannot be manipulated for records that are loaded in beforeLoad scripts. If you attempt to update a record loaded in beforeLoad, the logic is ignored. Data can be manipulated for records created in beforeLoad user events. Attaching a child custom record to its parent or detaching a child custom record from its parent triggers an edit event. <p>This event can be used with the following <code>context.UserEventType</code> s:</p> <ul style="list-style-type: none"> create edit view copy print email quick view
Returns	void
Since	Version 2015 Release 2

Parameters

 **Note:** The `scriptContext` parameter is a JavaScript object.

Parameter	Type	Description	Since
<code>scriptContext.newRecord</code>	record.Record	The new record.	Version 2015 Release 2
<code>scriptContext.type</code>	string	The type of operation invoked by the event. The type can be any of the possible values for the context.UserEventType enum. Type arguments vary depending on whether the event occurs during a <code>beforeLoad</code> , <code>beforeSubmit</code> , or <code>afterSubmit</code> operation. This parameter allows the script to branch out to different logic depending on the operation type. For example, a script with deltrees logic that deletes a record and all of its child records should only be invoked when type equals <code>delete</code> . User event scripts should always check the value of the type argument to avoid indiscriminate execution.	Version 2015 Release 2
<code>scriptContext.form</code>	serverWidget.Form	The current form.	Version 2015 Release 2
<code>scriptContext.request</code>	http.ServerRequest	The HTTP request information sent by the browser. If the event was triggered by a server action, this value is not present.	Version 2015 Release 2

beforeSubmit(scriptContext)

Description	<p>Executed prior to any write operation on the record. Changes made to the current record in this script persist after the write operation. This event can be used to validate the submitted record, perform any restriction and permission checks, and perform any last-minute changes to the current record.</p> <p>Notes:</p> <ul style="list-style-type: none"> ■ The approve, cancel, and reject argument types are only available for record types such as sales orders, expense reports, timebills, purchase orders, and return authorizations. ■ Only <code>beforeLoad</code> and <code>afterSubmit</code> user event scripts execute on the Message record type when a message is created by an inbound email case capture. Scripts set to execute on a <code>beforeSubmit</code> event do not execute. ■ User Event Scripts cannot override custom field permissions. For instance, if a user's role permissions and a custom field's permissions differ, <code>beforeSubmit</code> cannot update the custom field, even if the script is set to execute as Administrator. ■ Attaching a child custom record to its parent or detaching a child custom record from its parent triggers an edit event. <p>Best practices: To set a field on a record or make any changes to a record that is being submitted, do so on a <code>beforeSubmit</code> operation, not an <code>afterSubmit</code> operation. If you set a field on an <code>afterSubmit</code>, you duplicate a record whose data has already been committed to the database. This event can be used with the following context.UserEventType s:</p>
--------------------	--

	<ul style="list-style-type: none"> create edit xedit (inline editing) delete approve (only available for certain record types) cancel (only available for certain record types) reject (only available for certain record types) pack (only available for certain record types, for example Item Fulfillment records) ship (only available for certain record types, for example Item Fulfillment records) markcomplete (specify this type for a beforeSubmit script to execute when users click the Mark Complete link on call and task records) reassign (specify this type for a beforeSubmit script to execute when users click the Grab link on case records) editforecast (specify this type for a beforeSubmit script to execute when users update opportunity and estimate records using the Forecast Editor)
Returns	void
Since	Version 2015 Release 2

Parameters

 **Note:** The `scriptContext` parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>scriptContext.newRecord</code>	record.Record	required	The new record.	Version 2015 Release 2
<code>scriptContext.oldRecord</code>	record.Record	required	The old record.	Version 2015 Release 2
<code>scriptContext.type</code>	string	required	The trigger type. Use the context.UserEventType enum to set this value.	Version 2015 Release 2

context.UserEventType

Enum Description	Holds the string values for user event execution contexts. See the following for the user events types that are supported for user event entry points: <ul style="list-style-type: none"> <code>afterSubmit(scriptContext)</code> <code>beforeLoad(scriptContext)</code> <code>beforeSubmit(scriptContext)</code>
Module	SuiteScript 2.0 User Event Script Type
Since	Version 2015 Release 2

Values

- APPROVE

- CANCEL
- CHANGEPASSWORD



Note: The CHANGEPASSWORD user event is triggered when a user updates their NetSuite password in the NetSuite UI. The password change event is only visible in the NetSuite account in which the password was changed.

- COPY
- CREATE
- DELETE
- DROPSHIP
- EDIT
- EDITFORECAST
- EMAIL
- MARKCOMPLETE
- ORDERITEMS
- PACK
- PAYBILLS
- PRINT
- QUICKVIEW
- REASSIGN
- REJECT
- SHIP
- SPECIALORDER
- TRANSFORM
- VIEW
- XEDIT



Note: Attaching a child custom record to its parent or detaching a child custom record from its parent triggers an edit event.

SuiteScript 2.0 Workflow Action Script Type

Workflow action scripts allow you to create custom Workflow Actions that are defined on a record in a workflow. Workflow action scripts are useful for performing actions on sublists because sublist fields are not currently available through the Workflow Manager. Workflow action scripts are also useful when you need to create custom actions that execute complex computational logic that is beyond what can be implemented with the built-in actions.

For information about SuiteFlow workflows, see the following topics:

- [SuiteFlow Overview](#)
- [Working with Workflows](#)
- [SuiteFlow Reference and Examples](#)

For information about scripting with workflow action scripts, see [SuiteScript 2.0 Workflow Action Script Entry Points and API](#).

Workflow Action Script Sample

This sample shows how to store a return value from a custom action script into a workflow field. This example can be useful to satisfy the following use cases:

- You want to get a value from the Item sublist and use this value as a condition in the workflow. You use `obj.getSublistValue` in the script and return this in the workflow.
- You want to check if a certain item is existing in the Item sublist. The script returns "0" if item is not existing and "1" if it does.
- You want to make sure that all items in the Item sublist have a quantity equal to or greater than 1 (similar case as #2).

This script sample assumes the following set-up in the NetSuite account:

- Make sure that the script returns a value. You can specify this on the Parameters tab of the Script record page.
- In SuiteFlow, create a workflow field. The field should be the of the same type as the return parameter of the Workflow Action script.
- Within a state, add the custom action (this is the Workflow Action script).
- Add the return value from the Workflow Action script to the Store Result In field. This field is found in the custom action's Parameters. See the help topic [Storing a Return Value from a Custom Action Script in a Workflow Field](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType WorkflowActionScript
 */
define([],
function() {
    function onAction(scriptContext) {
        log.debug({
            title: 'Start Script'
        });
        var newRecord = scriptContext.newRecord;
        var itemCount = newRecord.getLineCount({
            sublistId: 'item'
        });

        log.debug({
            title: 'Item Count',
            details: itemCount
        });
        for(var i = 0; i < itemCount; i++)
        {
            var quantity = newRecord.getSublistValue({
                sublistId: 'item',
                fieldId: 'quantity',
                line: i
            });

            log.debug({
                title: 'Quantity of Item ' + i,
                details: quantity
            });
        }
    }
});
```

```

    });
    if(quantity === 0)
    {
        return 0;
    }
}
log.debug({
    title: 'End Script'
});
return 1;
}
return {
    onAction: onAction
}
});

```

SuiteScript 2.0 Workflow Action Script Entry Points and API

[onAction\(scriptContext\)](#)

onAction(scriptContext)

Description	Defines a Workflow Action script trigger point.
Returns	void
Since	Version 2016 Release 1

Parameters

 **Note:** The `scriptContext` parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>scriptContext.newRecord</code>	<code>record.Record</code>	required	The new record. <code>record.Record.save()</code> is not permitted.	2016.1
<code>scriptContext.oldRecord</code>	<code>record.Record</code>	required	The old record. <code>record.Record.save()</code> is not permitted.	2016.1
<code>scriptContext.form</code>	<code>serverWidget.Form</code>	optional	The form through which the script interacts with the record. This parameter is available only in the <code>beforeLoad</code> context.	2016.2
<code>scriptContext.type</code>	string	optional	An event type, such as create, edit, view, or delete.	2016.2
<code>scriptContext.workflowId</code>	integer	optional	The internal ID of the workflow that calls the script.	2016.2

SuiteScript 2.0 Global Objects

SuiteScript 2.0 includes the following global objects. You can use these objects in your scripts without loading them as dependencies.

- [define Object](#)
- [require Function](#)
- [log Object](#)
- [util Object](#)
- [toString\(\)](#)
- [JSON object](#)
- [Promise Object](#)

Note: In JavaScript, all functions are objects. The [define Object](#) [define Object](#) and [require Function](#) topics discuss the `define()` and `require()` functions used by SuiteScript 2.0 to load and define modules.

define Object

The `define` object is an overloaded function that is used to create entry point scripts and custom modules in SuiteScript 2.0. This function executes asynchronously on the client side and synchronously on the server side. The `define` object conforms to the [Asynchronous Module Definition \(AMD\)](#) specification.

Note: An overloaded function has multiple signatures. A signature is the function name and all available parameters.

SuiteScript 2.0 supports three `define()` signatures:

Type	Name	Return Type / Value Type	Description
Function	<code>define(moduleObject)</code>	object	Returns a module object based on the supplied <code>moduleobject</code> argument. The <code>moduleobject</code> argument can be any JavaScript object, including a function. Use this <code>define()</code> signature if your entry point script or custom module requires no dependencies.
	<code>define(id, [dependencies,] moduleObject)</code>	object	Loads all dependencies and then executes the supplied callback function. Returns a module object based on the callback.

Use the `define()` function to do the following:

- Create a SuiteScript script file. Load the required dependent modules and define the functionality for the SuiteScript script type in the callback function. The return statement in the callback function must include at least one entry point and entry point function. All entry points must belong to the same script type.

Any implementation of a SuiteScript script type that returns an entry point must use the `define()` Function.

- Create and return a custom module. You can then include the custom module as dependency in another script. Use the *`define([dependencies,] callback)`* signature if your module requires dependencies. If the custom module does not require any dependencies, use the *`define(moduleObject)`* signature.

For more information about custom modules, see [SuiteScript 2.0 Custom Modules](#).

For more information about entry points, see [SuiteScript 2.0 Script Types](#).

define() Function Guidelines

Use the following guidelines with the `define()` Function:

- SuiteScript API calls can be executed only after the `define` callback's return statement has executed. Consequently, you cannot use native SuiteScript 2.0 module methods when you *create* a custom module. You can make SuiteScript API calls after the Module Loader creates and loads the custom module.
- If you need to debug your code on demand in the NetSuite Debugger, you must use a `require()` Function. The NetSuite Debugger cannot step through a `define()` Function.
- Any dependencies used in the `define()` Function are loaded before the callback function executes.
- You can load only modules that are stored in the NetSuite file cabinet. Do not attempt to import scripts via HTTP/S.

For example, if given `define(['http://somewebsite.com/purchaseTotal.js'], function(purchaseTotal) { ... }) ;`, the `purchaseTotal` dependency is not valid.

define(moduleObject)

Description	Function used to create entry point scripts and custom modules in SuiteScript 2.0. For more information, see SuiteScript 2.0 Entry Point Script Creation and Deployment and SuiteScript 2.0 Custom Modules . Use this <code>define()</code> signature if your entry point script or custom module requires no dependencies. If you are creating an entry point script, the <code>define()</code> function must return an object consisting of at least one key/value pair. Each key must be an entry point and the corresponding value must be a named entry point function. All entry points must be for the same script type. Your script can have only one entry point script and the entry point script must be only one script type.
Returns	Object
Global object	define Object
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description	Since
<code>moduleObject</code>	Object	Required	A callback function or a module object	Version 2015 Release 2

Syntax

The following code snippets show sample syntax for the `define(moduleobject)` function signature. These snippets are not functional examples or a complete list.

Define a Function

```
// lib.js
define({
  test: function ()
  {
    return true;
  }
});
```

OR

```
// lib.js
define(function ()
{
  return true
});
```

Define an object

```
// lib.js
define({
  color: "black",
  size: "unysize"
});
```

Define a Primitive Value

```
// lib.js
define("test");
```

`define(id, [dependencies,] moduleObject)`

Method Description	Function used to create entry point scripts and custom modules in SuiteScript 2.0. For more information, see SuiteScript 2.0 Entry Point Script Creation and Deployment and SuiteScript 2.0 Custom Modules . If you are creating an entry point script, the <code>define()</code> function must return an object consisting of at least one key/value pair. Each key must be an entry point and the corresponding value must be a named entry point function. All entry points must be for the same script type. Your script can have only one entry point script and the entry point script must be only one script type. Your entry point script can, however, load multiple custom modules as dependencies. There is no limit to the number of dependencies your entry point script can load.
Returns	Object
Global object	define Object
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description	Since
id	string	optional	Defines the id of the module	Version 2015 Release 2
dependencies	string []	optional	Represents any module dependencies required by the callback function. Use the following syntax: <ul style="list-style-type: none"> Native SuiteScript 2.0 modules: ['N/<module name>'] Custom modules: ['/'<path to module file in File Cabinet>/<module name>'] For other options, see Module Dependency Paths .	Version 2015 Release 2
moduleObject	Function Object	required	A callback function or a module object	Version 2015 Release 2

Errors

Error Code	Message	Thrown If
MODULE_DOES_NOT_EXIST	Module does not exist: {module path/name}	The NetSuite module or custom module dependency does not exist. If multiple modules do not exist, NetSuite only reports the first error encountered. If you receive this error, verify that all module paths and names are correct.

Syntax for Module ID

The following code snippet shows sample syntax for the `define(id, [dependencies,] callback)` function signature. It is not a functional example or complete list.

```
...
define('mymodule', ['/test', '/sample'], function(test, sample){...});
...
```

Syntax for Entry Point Script

The following code snippet shows a sample SuiteScript user event script type that creates a Phone Call record on the `afterSubmit` trigger.

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

define(['N/record'],
  function (record)
  {
    function createPhoneCall(context)
```

```

{
  if (context.type !== context.UserEventTypes.CREATE)
    return;
  var customerRecord = context.newRecord;
  if (customerRecord.getValue('salesrep'))
  {
    var call = record.create({
      type: record.Type.PHONE_CALL,
      isDynamic: true
    });
    call.setValue({
      fieldId: 'title',
      value: 'Make follow-up call to new customer'
    });
    call.setValue('assigned', customerRecord.getValue('salesrep'));
    call.setValue('phone', customerRecord.getValue('phone'));
    try
    {
      var callId = call.save();
      log.debug({
        title: 'Call record created successfully',
        details: 'Id: ' + callId
      });
    }
    catch (e)
    {
      log.error(e.name);
    }
  }
  return {
    afterSubmit: createPhoneCall
  };
});

```

Syntax for Custom Module

The following code snippets show the syntax for creating a custom SuiteScript 2.0 module in the script file `lib.js`.

```

// lib.js
define(['./api/bar'], function(bar){    // require bar custom module
  return {
    makeSomething: function(){          // define function lib.makeSomething()
      var barObj = bar.create();        // use create() function from bar custom module
      return bar.convertToThing();      // returns the value of bar module function convertToThing()
    }
  }
});

```

The following code snippet shows the syntax for calling the function `lib` from the custom module `test.js` in a separate script file:

```

// test.js
require(['lib'], function (lib) {    // require custom module (defined above)

```

```
return lib.makeSomething(); // return value of makeSomething function in custom module

});
```

require Function

The require Function is a global object that implements the require() Module Loader interface for SuiteScript 2.0. It conforms to the Asynchronous Module Definition (AMD) specification. When NetSuite executes the require() Function, it executes the callback function and loads the dependencies when they are needed.

This function executes asynchronously on the client side and synchronously on the server side.

Note: Only use the require() Function if you want to loading an existing module. If you want to create an entry point script or a new custom module, use the [define Object](#).

Use the require() Function to achieve progressive loading of native SuiteScript 2.0 modules and custom modules. When you use the require() Function, dependencies are not loaded until they are needed. This can help increase script performance.

For example, if you add `lib1` as a dependency. When you call a method that is part of `lib1`, the Module Loader loads the module and executes the method. See [Syntax](#).

Type	Name	Return Type / Value Type	Description
Function	require([dependencies,] callback)	Void	Loads a SuiteScript 2.0 entry point script or a SuiteScript 2.0 custom module. Executes the callback function and loads the dependencies when they are required.

Note: To configure a require Object, you can associate a script to a JSON configuration file using a JSDoc tag. This is helpful to configure loading of a custom module. Properties that can hold feature metadata, aliases, paths, package, and mapping information related to a module id are supported. See [require Configuration](#).

require([dependencies,] callback)

Method Description	<p>Function used to load a module only when the module is needed. When NetSuite executes the require() Function, it executes the callback function and loads the dependencies when they are required. If you add a module as a dependency and the module is never used, the dependency is never loaded.</p> <p>Note: This function conforms to the Asynchronous Module Definition (AMD) specification. For more information, see require Function.</p>
Returns	Void
Global object	require Function

Since	Version 2015 Release 2
-------	------------------------

Parameters

Parameter	Type	Required / Optional	Description	Since
dependencies	string []	Optional	Represents any module dependencies required by the callback function. Use the following syntax: <ul style="list-style-type: none"> Native SuiteScript 2.0 modules: ['N/<module name>'] Custom modules: ['/'<path to module file in File Cabinet>/<module name>'] See Module Dependency Paths .	Version 2015 Release 2
callback	Function	Required	Callback function to execute. Dependent modules are not loaded until they are required.	Version 2015 Release 2

Errors

Error Code	Message	Thrown If
MODULE_DOES_NOT_EXIST	Module does not exist: {module path/name}	The NetSuite module or custom module dependency does not exist. If multiple modules do not exist, NetSuite only reports the first error encountered. If you receive this error, verify that all module paths and names are correct.

Syntax

The following example shows progressive loading of modules in a script. For a functional example, see [require Function](#).

```
define({
  newInstance: function (type)
  {
    switch (type)
    {
      case 'lib1' :
        require(['lib1'], function (lib1) // Module Loader loads lib1
        {
          return new lib1();
        })
        break;
      case 'lib2' :
        require(['lib2'], function (lib2) // Module Loader loads lib2
        {
          return new lib2();
        })
        break;
      default :
        return null;
    }
  }
});
```

```

    }
  });

```

require Configuration

SuiteScript provides advanced options that provide you with greater control over require configuration.

If you set up a valid `@NAmDConfig` JSDoc tag, SuiteScript implements the require configuration settings *before* loading dependencies. Configure the require Object before loading dependencies so that you can run multiple client scripts with different configurations. Using the JSDoc tag can also support re-use by letting you use a common configuration across multiple scripts.

To configure a require Object, do the following:

- Add the `@NAmDConfig` tag and provide a file cabinet path to the configuration file

```

/**
 * @NAmDConfig /SuiteScripts/configuration.json
 */

```

- SuiteScript will require a custom entry point module and its dependencies using the AMD configuration. For a list of supported configuration parameters, see [require Configuration Parameters](#). Your require configuration must be in JSON format. For example:

```

{
  "baseUrl" : "/SuiteBundles"
}

```



Important: Ensure that configuration file uses JSON syntax (and not JavaScript syntax). For more information about JSON, visit <http://json.org/>.

You can use `JSON.stringify(obj)` to convert a JavaScript object value to a key-value pair string in JSON form.

require Configuration Parameters

SuiteScript accepts the configuration values outlined at <https://github.com/amdjs/amdjs-api/blob/master/CommonConfig.md> (version fd45c71).

You can use the JS Doc tag to point a configuration file that holds the configuration values, such as when you want to set properties before loading a custom module, or set up configuration for improved compatibility.

The following configuration parameters are supported for require Object configuration:

Parameter	Type	Required / Optional	Sample Usage	Since
baseUrl	string	Optional	<ul style="list-style-type: none"> ■ To configure a shorter relative path by indicating the root folder that holds the modules in the file cabinet. 	Version 2015 Release 2
paths	Object	Optional	<ul style="list-style-type: none"> ■ To create a named alias to a path ■ For testing purposes, pass in an object that serves as a mock-up of another module. ■ To set up a custom name for a SuiteScript module 	Version 2015 Release 2

Parameter	Type	Required / Optional	Sample Usage	Since
packages	Object[]	Optional	<ul style="list-style-type: none"> To configure a special lookup suitable for traditional CommonJS packages that you want to use as a custom module. 	Version 2015 Release 2
map	Object	Optional	<ul style="list-style-type: none"> To specify an alias. To handle multiple names for a module To load a set of identically-named but unique modules, such as dependency on multiple module versions. 	Version 2015 Release 2
config	Object	Optional	<ul style="list-style-type: none"> To assign attributes, such as metadata. 	Version 2015 Release 2
shim	Object	Optional	<ul style="list-style-type: none"> To prepare a non-AMD JS library for loading. 	Version 2015 Release 2

require.config()

Configuration of a require Object is optional and for advanced usage only. If you must configure a require Object, the `@NAmDConfig` tag is suited for general use and is the preferred way to configure a require Object. However, existing scripts with calls to `require.config` can use this method with a context argument (although not recommended). Ensure that the call includes a context parameter and that its value is not a file path.

log Object

The log Object is loaded by default by NetSuite for all script types. You do not need to load it manually. However, you can choose to load it via [N/log Module](#), such as for testing purposes.

log Object Members

- log.debug(options)
- log.audit(options)
- log.emergency(options)
- log.error(options)

For more details about the log Object and its methods, see [N/log Module](#).

util Object

The util Object is loaded by default by NetSuite for all script types. You do not need to load it manually. However, you can choose to load it via [N/util Module](#), such as for testing purposes.

util Object Members

- util.isArray(obj)

- `util.isBoolean(obj)`
- `util.isDate(obj)`
- `util.isFunction(obj)`
- `util.isNumber(obj)`
- `util.isObject(obj)`
- `util.isRegExp(obj)`
- `util.isString(obj)`

The util object also includes the following utility methods:

- `util.each(iterable, callback)`
- `util.extend(receiver, contributor)`
- `util.nanoTime()`

For more details about the util Object and its methods, see [N/util Module](#).

toString()

Method Description	<p>Method used to determine an object's type. This is a global method that is loaded by default for all native SuiteScript 2.0 API objects.</p> <div>  Note: Consider this method a replacement for the <code>instanceOf</code> operator (which is not supported). SuiteScript 2.0 members are immutable; you cannot construct or modify a native SuiteScript 2.0 member. Consequently, if you attempt to call <code>instanceOf</code>, an undefined error is thrown. </div> <p>For information about <code>toString()</code> as a native JavaScript method, see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/object/toString.</p>
Returns	The object type as a string
Supported Script Types	All script types
Governance	None
Since	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example.

```
...
var type = mapContext.toString();    // When called on mapReduce.MapContext, toString returns "mapReduce.MapContext"
...
```

JSON object

SuiteScript 2.0 supports the JavaScript Object Notation (JSON) standard. You can use the JSON object to parse text as a JSON object and convert strings to JSON notation. For more information, see [JSON.parse\(text\)](#) and [JSON.stringify\(obj\)](#).



Important: The following sections are included as a summary and are intended for reference only. For additional information about JSON, see <http://www.ietf.org/rfc/rfc4627.txt>.

JSON.parse(text)

Method Description	Parse a string as a JSON object and returns the object. The text parameter must conform to the JSON standard. See http://www.ietf.org/rfc/rfc4627.txt .
Returns	Object
Supported Script Types	All script types
Governance	None
Global object	JSON object
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description	Since
text	string	Required	Text to parse as a JSON object. The string must conform to the JSON standard.	Version 2015 Release 2
reviver	Function	Optional	Specifies how to transform the parsed value before it is returned	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example.

```
...
var text = '{ "employees" : [' +
    '{ "firstName":"John" , "lastName":"Doe" },' +
    '{ "firstName":"Anna" , "lastName":"Smith" },' +
    '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
var obj = JSON.parse(text);
var firstEmp = obj.employees[1].firstName + " " + obj.employees[1].lastName;
...
```

JSON.stringify(obj)

Method Description	Converts a JavaScript object value to a key-value pair string in the JSON format. For more information about JSON object format, see http://www.ietf.org/rfc/rfc4627.txt .
Returns	Object
Supported Script Types	All script types
Governance	None

Global object	JSON object
Since	Version 2015 Release 2

Parameters

Parameter	Type	Required / Optional	Description	Since
text	string	Required	Text to parse as a JSON object. The string must conform to the JSON standard.	Version 2015 Release 2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example.

```
var contact = {
    firstName: 'John',
    lastName : 'Doe',
    jobTitle : 'CEO'
};

var jsonString = JSON.stringify(contact);
```

This method converts the `contact` object to the following string:

```
{"firstName":"John","lastName":"Doe","jobTitle":"CEO"}
```

Promise Object

In SuiteScript 2.0, all client scripts support the use of Promises. With Promises, developers can write asynchronous code that is intuitive and efficient. SuiteScript 2.0 provides promise APIs for selected modules (see [SuiteScript 2.0 Promise APIs](#)). In addition, you can create custom Promises in all client scripts (see [Custom Promises](#)).

A promise is a JavaScript object that represents the eventual result of an asynchronous process. After this object is created, it serves as a placeholder for the future success or failure of the operation. During the period of time that the promise object is waiting, the remaining segments of the script can execute.

A Promise holds one of the following values:

- `fulfilled` – The operation is successful.
- `rejected` – The operation failed.
- `pending` – The operation is still in progress and has not yet been fulfilled or rejected.

When it is first created, a Promise holds the value `pending`. After the associated process is complete (from success or failure), the value changes to `fulfilled` or `rejected`. A success or failure callback function attached to the Promise is called when the process is complete. Note that a Promise can only succeed or fail one time. When the value of the Promise updates to fulfilled or rejected, it cannot change.

For additional information regarding Promises, see <https://www.promisejs.org/>.

SuiteScript 2.0 Promise APIs

SuiteScript 2.0 provides client-side promise APIs. For supported modules members and additional API information, see [SuiteScript 2.0 Modules](#).



Important: Although these modules as a whole are supported in client and server-side scripts, their promise APIs are supported only in client scripts.

The available promise APIs are named so that they correspond with their synchronous counterparts. The distinction is that the promise APIs have names that are suffixed with `.promise`. For example, the `search.create(options)` API has a promise version named `search.create.promise(options)`.

The following is a basic example of how to use a promise API in a client script.

```
/**
 * @NAPIVersion 2.0
 */
define(['N/search'],
function(search)
{
    function doSomething()
    {
        search.create.promise({
            type: 'salesorder'
        })
        .then(function(result) {
            log.debug("Completed: " + result);
            //do something after completion
        })
        .catch(function(reason) {
            log.debug("Failed: " + reason)
            //do something on failure
        });
    }
    return
    {
        pageInit: doSomething
    }
});
```

This example demonstrates how to chain promises created with promise APIs.

```
/**
 * @NAPIVersion 2.0
 */
define(['N/search'],
function(search)
{
    function doSomething()
    {
        var filter = search.createFilter({
            name: 'mainline',
            operator: search.Operator.IS,
            values:['T']
        });
```

```

    });
    search.create.promise({
        type: 'salesorder',
        filters:[filter]
    })
    .then(function(searchObj) {
        return searchObj.run().each.promise(
            function(result, index){
                //do something
            })
    })
    .then(function(result) {
        log.debug("Completed: " + result)
        //do something after completion
    })
    .catch(function(reason) {
        log.debug("Failed: " + reason)
        //do something on failure
    });
})
return
{
    pageInit: doSomething
}
}
);

```

Custom Promises

The following example shows a custom Promise. Custom Promises do not utilize the SuiteScript 2.0 promise APIs.

```

/**
 * @NAPIVersion 2.0
 */
define(function(){
    function doSomething(addresses){
        var promise = new Promise(function(resolve, reject){
            var url = 'https://your.favorite.maps/api/directions?start=' + addresses.start + '&end=' + addresses.end,
                isAsync = true,
                xhr = new XMLHttpRequest();

            xhr.addEventListener('load', function (event) {
                if (xhr.readyState === 4) {
                    if (xhr.status === 200) {
                        resolve(xhr.responseText );
                    }
                    else {
                        reject( xhr.statusText );
                    }
                }
            });
        });
        xhr.addEventListener('error', function (event) {
            reject( xhr.statusText );
        });
    }
});

```

```
        });  
        xhr.open('GET', url, isAsync);  
        xhr.send();  
    });  
  
    return promise;  
}  
  
return {  
    lookupDirections: doSomething  
};  
});
```

SuiteScript 2.0 Modules

SuiteScript 2.0 APIs are organized into various modules, based on behavior. These modules are described below.

Note: As a best practice, you should load only the modules that are needed by your script. However, you can load all SuiteScript 2.0 modules at one time. Do this by passing the modules' parent directory to the `define()` statement and its callback function: `define(['N'], function(N) { ... })`; . This is a convenient way to load all modules, but does sacrifice the performance advantage of loading only the modules that are needed. We provide this feature so that you can test and familiarize yourself with SuiteScript 2.0. We do not recommend that you load all modules at once in a production environment.

Module	Description
N/action Module	Load the N/action module APIs to execute business logic to update the state of a record. Action APIs emulate NetSuite UI buttons.
N/auth Module	Load the auth module when you want to change your NetSuite login credentials.
N/cache Module	Load the cache module to enable the caching of needed data and improve performance.
N/config Module	Load the config module when you want to access NetSuite configuration settings. See config.Type for a list of supported configuration pages.
N/crypto Module	Load the crypto module to work with hashing, hash-based message authentication (hmac), and symmetrical encryption. You can access a set of wrappers for OpenSSL's hash, hmac, cipher, and decipher methods.
N/currency Module	Load the currency module to work with exchange rates within your NetSuite account. You can use the currency module to find the exchange rate between two currencies based on a certain date.
N/currentRecord Module	Load the currentRecord module to access the record instance that you are currently working on. You can then use the record instance in a client-side context.
N/email Module	Load the email module when you want to send email messages from within NetSuite. You can use the email module to send regular, bulk, and campaign email.
N/encode Module	Load the encode module when you want to convert a string to another type of encoding. See encode.Encoding for a list of supported character set encoding.
N/error Module	Load the error module when you want to create your own custom SuiteScript errors. Use these custom errors in try-catch statements to abort script execution.
N/file Module	Load the file module to work with files in NetSuite.
N/format Module	Load the format module to convert strings into a specified format and to parse formatted data into strings.
N/http Module	Load the http module to make http calls. All HTTP content types are supported.
N/https Module	Load the https module to make https calls. You can also use this module to encode binary content or securely access a handle to the value in a NetSuite credential field.
N/log Module	Load the log module when you want to access methods for logging script execution details. Module members are also supported by the global log Object .
N/plugin Module	Load the plugin module to load custom plug-in implementations.
N/portlet Module	Load the portlet module when you want to resize or refresh a form portlet.

Module	Description
N/query Module	Load the query module to create and run searches using the SuiteAnalytics Workbook query engine.
N/record Module	Load the record module to work with NetSuite records.
N/redirect Module	Load the redirect module when you want to redirect users to one of the following: <ul style="list-style-type: none"> URL Suitelet Record Task link Saved search Unsaved search
N/render Module	Load the render module to create forms or email from templates and to print to PDF or HTML.
N/runtime Module	Load the runtime module when you want to access the runtime settings for company, script, session, system, user, or version.
N/search Module	Load the search module to create and run on demand or saved searches and analyze and iterate through the search results. You can search for a single record by keywords, create saved searches, search for duplicate records, or return a set of records that match filters you define.
N/sftp Module	Load the sftp module to connect to a remote FTP server via SFTP and transfer files.
N/sso Module	Load the sso module when you want to generate outbound single sign-on (SuiteSignOn) tokens
N/task Module	Load the task module to create tasks and place them in the internal NetSuite scheduling or task queue. Use the task module to schedule scripts, run Map/Reduce scripts, import CSV files, merge duplicate records, and execute asynchronous workflows.
N/transaction Module	Load the transaction module to void transactions.
N/ui/dialog Module	Load the dialog module to create a modal dialog that persists until a button on the dialog is pressed.
N/ui/message module	Load the message module to display a message at the top of the screen under the menu bar.
N/ui/serverWidget Module	Load the serverWidget module when you want to work with the user interface within NetSuite.
N/url Module	Load the url module when you want to determine URL navigation paths within NetSuite or format URL strings.
N/util Module	Load the util module when you want to manually access util methods. Module members are also supported by the global <code>util</code> Object.
N/workflow Module	Load the workflow module to initiate new workflow instances or trigger existing workflow instances.
N/xml Module	Load the xml module to validate, parse, read, and modify XML documents.

N/action Module

The N/action module APIs let you execute business logic to update the state of records in view mode. To execute business logic on records that you are editing, use the record macro APIs, included in the [N/record Module](#) module. See [Record Object Members](#) and [Macro Object Members](#). Action and Macro APIs are the programmatic equivalent to clicking a button in the UI. To learn more, see [Overview of Record Action and Macro APIs](#).

The changes that you make to records with N/action module APIs are persisted in the database immediately. For example, consider the timebill record. After you click the **Approve** button in the UI, the timebill and its entries are saved in an approved state, and this change is immediately updated in the database.

Governance for action module APIs varies for actions and record types. See the action help for governance information specific to actions and record types.

A limited number of individual actions for specific record types are supported. For details, see [Supported Record Actions](#).

 **Note:** For supported script types, see individual member topics listed below.

- [N/action Module Members](#)
- [Action Object Members](#)
- [N/action Module Script Sample](#)

N/action Module Members

Member Type	Name	Return Type / Value Type	Description
Object	action.Action	Object	Encapsulates a NetSuite record action.
	Plain JavaScript Object	Object	A plain JavaScript object of actions available for a record type.
Method	action.get(options)	action.Action	Returns an executable record action for the given record type.
	action.get.promise(options)	Promise	Asynchronously returns an executable record action for the given record type.
	action.find(options)	Object	Returns a plain JavaScript object of available record actions for the given record type.
	action.find.promise(options)	Promise	Asynchronously returns a plain JavaScript object of available record actions for the given record type.
	action.execute(options)	Object	Executes the record action and returns action results in an object.
	action.execute.promise(options)	Promise	Asynchronously executes the record action and returns the action results in an object.

Action Object Members

The following members are called on [action.Action](#).

Member Type	Name	Return Type/ Value Type	Description
Method	Action.execute(options)	Object	Executes the action and returns the action results in an object.
	Action.execute.promise(options)	Promise	Executes the action asynchronously and returns the action results in an object.
	Action(options)	Object	Executes the action and returns the action results in an object.
	Action.promise(options)	Promise	Executes the action asynchronously and returns the action results in an object.
Property	Action.id	string	The ID of the action. For a list of action IDs, see Supported Record Actions .
	Action.recordType	string	The type of the record on which the action is to be performed. For a list of record types, see record.Type .
	Action.label	string	The action label.
	Action.description	string	The action description.
	Action.parameters	Object	The action parameters.

N/action Module Script Sample

These samples use the `require` function, so that you can copy each script into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For more information, see [SuiteScript 2.0 Script Architecture](#) and [SuiteScript 2.0 Script Types](#).



Important: The samples included in this section are intended to show how actions work in SuiteScript at a high-level. For specific samples, see [Supported Record Actions](#).

The following server-side script sample finds and executes an action on the timebill record without promises.

```
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

```

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

```

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

```

action.Action

Object Description	Encapsulates a NetSuite record action. This object is returned by the action.get(options) and action.find(options) methods.
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Methods and Properties	Action Object Members

Since	2018.2
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var action = actionMod.get({recordType: 'timebill', id: 'approve'});
...
// Add additional code
```

Action.execute(options)

Method Description	Executes the action and returns the action result in a object. The <code>response</code> property of the result object holds the actual response returned by the action implementation. The <code>notifications</code> property of the result object is an array of notification objects. It contains the details of errors and warnings that occurred during action execution. If the action executes successfully, the <code>notifications</code> property is usually empty. If the Action object is qualified (it is a result of an <code>action.get()</code> or <code>action.find()</code> call that provides the <code>recordId</code>), then it is not required to provide a <code>recordId</code> and the <code>options.params.recordId</code> parameter is optional. If <code>options.params.recordId</code> is provided during execution, it takes precedence over the <code>recordId</code> stored in the Action object.
Returns	Object
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Parent Object	action.Action
Sibling Object Members	Action Object Members
Since	2018.2

Parameters



Note: The parameters that are required vary for action types. The only parameter that is always required is `options.recordId`, unless the action object is qualified. An action object is qualified if it is the result of an `action.get()` or `action.find()` call that provides the `recordId`.

Parameter	Type	Required / Optional	Description
<code>options.Object</code>	Object	required or optional	The parameters that need to be provided depend on the action implementation. See the action help.
<code>options.params.recordId</code>	string	required or optional	The record instance ID of the record on which the action is to be performed.

Parameter	Type	Required / Optional	Description
			This is the NetSuite record internal ID.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var result = action.execute({recordId: 1});
...
// Add additional code
```

Action.execute.promise(options)

Method Description	Executes the action asynchronously and returns the action result in a plain JavaScript object. The action result is returned in an object. The <code>response</code> property of the results object shows the action result. If the action fails, it is listed in the results object's <code>notifications</code> property. If the action executes successfully, the <code>notifications</code> property is usually empty. If the Action object is qualified (it is a result of an <code>action.get()</code> or <code>action.find()</code> call that provides the <code>recordId</code>), then it is not required to provide a <code>recordId</code> and the <code>options.params.recordId</code> parameter is optional. If <code>options.params.recordId</code> is provided during execution, it takes precedence over the <code>recordId</code> stored in the Action object.
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Parent Object	action.Action
Sibling Object Members	Action Object Members
Since	2018.2

Parameters



Note: The parameters that are required vary for action types. The only parameter that is always required is `options.recordId`, unless the action object is qualified. An action object is qualified if it is the result of an `action.get()` or `action.find()` call that provides the `recordId`.

Parameter	Type	Required / Optional	Description
<code>options.Object</code>	Object	required or optional	The parameters that need to be provided depend on the action implementation. See the action help.
<code>options.params.recordId</code>	string	required or optional	The record instance ID of the record on which the action is to be performed.

Parameter	Type	Required / Optional	Description
			This is the NetSuite record internal ID.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
action.execute.promise({recordId: 1}).then(function(result) { /* process result here */ });
...
// Add additional code
```

Action(options)

Method Description	<p>Executes the action and returns the action result in a plain JavaScript object. The action result is returned in an object. The <code>response</code> property of the results object shows the action result. If the action fails, it is listed in the results object's <code>notifications</code> property. If the action executes successfully, the <code>notifications</code> property is usually empty. If the Action object is qualified (it is a result of an <code>action.get()</code> or <code>action.find()</code> call that provides the <code>recordId</code>), then it is not required to provide a <code>recordId</code> and the <code>options.params.recordId</code> parameter is optional. If <code>options.params.recordId</code> is provided during execution, it takes precedence over the <code>recordId</code> stored in the Action object.</p> <div> Note: Replace Action with the name of the action you are executing. </div>
Returns	Object
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Parent Object	action.Action
Sibling Object Members	Action Object Members
Since	2018.2

Parameters



Note: The parameters that are required vary for action types. The only parameter that is always required is `options.recordId`, unless the action object is qualified. An action object is qualified if it is the result of an `action.get()` or `action.find()` call that provides the `recordId`.

Parameter	Type	Required / Optional	Description
<code>options.Object</code>	Object	required or optional	The parameters that need to be provided depend on the action implementation. See the action help.
<code>options.params.recordId</code>	string	required or optional	The record instance ID of the record on which the action is to be performed.

Parameter	Type	Required / Optional	Description
			This is the NetSuite record internal ID.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var result = action({recordId: 1});
...
// Add additional code
```

Action.promise(options)

Method Description	<p>Executes the action asynchronously and returns the action result in a plain JavaScript object. The action result is returned in an object. The <code>response</code> property of the results object shows the action result. If the action fails, it is listed in the results object's <code>notifications</code> property. If the action executes successfully, the <code>notifications</code> property is usually empty. If the Action object is qualified (it is a result of an <code>action.get()</code> or <code>action.find()</code> call that provides the <code>recordId</code>), then it is not required to provide a <code>recordId</code> and the <code>options.params.recordId</code> parameter is optional. If <code>options.params.recordId</code> is provided during execution, it takes precedence over the <code>recordId</code> stored in the Action object.</p> <div> Note: Replace Action with the name of the action you are executing. </div>
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Parent Object	action.Action
Sibling Object Members	Action Object Members
Since	2018.2

Parameters



Note: The parameters that are required vary for action types. The only parameter that is always required is `options.recordId`, unless the action object is qualified. An action object is qualified if it is the result of an `action.get()` or `action.find()` call that provides the `recordId`.

Parameter	Type	Required / Optional	Description
<code>options.Object</code>	Object	required or optional	The parameters that need to be provided depend on the action implementation. See the action help.
<code>options.params.recordId</code>	string	required or optional	The record instance ID of the record on which the action is to be performed.

Parameter	Type	Required / Optional	Description
			This is the NetSuite record internal ID.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
action.promise({recordId: 1}).then(function(result) { /* process result here */ });
...
// Add additional code
```

Action.id

Property Description	The ID of the action. For a list of action IDs, see Supported Record Actions .
Type	string
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var id = action.id; // get the id of the action
...
// Add additional code
```

Action.recordType

Property Description	The type of the record on which the action is to be performed. For a list of record types, see record.Type .
Type	string
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module

Sibling Object Members	Action Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var recordType = action.recordType; // get the record type
...
// Add additional code
```

Action.label

Property Description	The action label.
Type	string
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var label = action.label; // get the action label
...
// Add additional code
```

Action.description

Property Description	The action description.
Type	string
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var description = action.description; // get the action description
...
// Add additional code
```

Action.parameters

Property Description	The action parameters.
Type	Object
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var params = action.parameters; // get the action parameters
...
// Add additional code
```

action.get(options)

Method Description	Returns an executable record action for the specified record type. If the <code>recordId</code> parameter is specified, the action object is returned only if the specified action can be executed on the specified record instance.
Returns	action.Action
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.recordType</code>	string	required	The record type. For a list of record types, see record.Type .
<code>options.recordId</code>	string	optional	The record instance ID.
<code>options.id</code>	string	required	The ID of the action. For a list of action IDs, see Supported Record Actions .

Errors

Error Code	Thrown If
SSS_INVALID_RECORD_TYPE	The specified record type is invalid.
SSS_MISSING_REQD_ARGUMENT	A required parameter is missing.
SSS_INVALID_ACTION_ID	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
RECORD_DOES_NOT_EXIST	The specified record instance does not exist.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var action = actionMod.get({recordType: 'timebill', id: 'approve'});
...
// Add additional code
```

action.get.promise(options)

Method Description	Returns an executable record action for the specified record type asynchronously. If the <code>recordId</code> parameter is specified, the action object is returned only if the specified action can be executed on the specified record instance.
Returns	Promise
Supported Script Types	Client scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module

Sibling Object Members	Action Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.recordType</code>	string	required	The record type. For a list of record types, see record.Type .
<code>options.recordId</code>	string	optional	The record instance ID.
<code>options.id</code>	string	required	The ID of the action. For a list of action IDs, see Supported Record Actions .

Errors

Error Code	Thrown If
SSS_INVALID_RECORD_TYPE	The specified record type is invalid.
SSS_MISSING_REQD_ARGUMENT	A required parameter is missing.
SSS_INVALID_ACTION_ID	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
RECORD_DOES_NOT_EXIST	The specified record instance does not exist.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
actionMod.get.promise({recordType: 'timebill', id: 'approve'}).then(function(action) {
    // do something with the action object
});
...
// Add additional code
```

action.find(options)

Method Description	Performs a search for available record actions. If only the <code>recordType</code> parameter is specified, all actions available for the record type are returned. If the <code>recordId</code> parameter is also
---------------------------	--

	<p>specified, then only actions that qualify for execution on the given record instance are returned. If the <code>id</code> parameter is specified, then only the action with the specified action ID is returned.</p> <p>This method returns a plain JavaScript object of NetSuite record actions available for the record type. The object contains one or more action.Action objects. If there are no available actions for the specified record type, an empty object is returned.</p> <p>If the <code>recordId</code> is specified in this call, the actions that are found are considered qualified. You do not have to provide the <code>recordId</code> to execute a qualified action.</p>
Returns	Object
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.recordType</code>	string	required	The record type. For a list of record types, see record.Type .
<code>options.recordId</code>	string	optional	The record instance ID.
<code>options.id</code>	string	optional	The action ID.

Errors

Error Code	Thrown If
SSS_INVALID_RECORD_TYPE	The specified record type is invalid.
SSS_MISSING_REQD_ARGUMENT	The <code>options.recordType</code> parameter is missing or undefined.
SSS_INVALID_ACTION_ID	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
RECORD_DOES_NOT_EXIST	The specified record ID does not exist.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
```

```

...
var actions = action.find({
    recordType: 'timebill',
    recordId: recordId
});
...
// Add additional code

```

action.find.promise(options)

Method Description	<p>Performs a search for available record actions asynchronously. If only the <code>recordType</code> parameter is specified, all actions available for the record type are returned. If the <code>recordId</code> parameter is also specified, then only actions that qualify for execution on the given record instance are returned. If the <code>id</code> parameter is specified, the only the action with the specified action ID is returned.</p> <p>This method returns a plain JavaScript object of NetSuite record actions available for the record type. The object contains one or more action.Action objects. If there are no available actions for the specified record type, an empty object is returned.</p> <p>If the <code>recordId</code> is specified in this call, the actions that are found are considered qualified. You do not have to provide the <code>recordId</code> to execute a qualified action.</p>
Returns	Promise
Supported Script Types	Client scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.recordType</code>	string	required	The record type. For a list of record types, see record.Type .
<code>options.recordId</code>	string	optional	The record instance ID.
<code>options.id</code>	string	optional	The action ID.

Errors

Error Code	Thrown If
<code>SSS_INVALID_RECORD_TYPE</code>	The specified record type is invalid.
<code>SSS_MISSING_REQD_ARGUMENT</code>	The <code>options.recordType</code> parameter is missing or undefined.

Error Code	Thrown If
SSS_INVALID_ACTION_ID	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
RECORD_DOES_NOT_EXIST	The specified record ID does not exist.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var promise = action.find.promise({recordType: 'timebill'});
promise.then(function(actionList) {
    // do something with the list of actions
});
...
// Add additional code
```

action.execute(options)

Method Description	Executes the record action and returns the action results in a plain JavaScript object. If the action fails, it is listed in the results object's <code>notifications</code> property. If the action executes successfully, the <code>notifications</code> property is usually empty.
Returns	Object
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.recordType</code>	string	required	The record type. For a list of record types, see record.Type .
<code>options.id</code>	string	required	The action ID.

Parameter	Type	Required / Optional	Description
			For a list of action IDs, see Supported Record Actions .
<code>options.params</code>	Object	required	Action arguments.
<code>options.params.recordId</code>	string	required	The record instance ID. This is the NetSuite record internal ID.

Errors

Error Code	Thrown If
<code>SSS_INVALID_RECORD_TYPE</code>	The specified record type is invalid.
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required parameter is missing.
<code>SSS_INVALID_ACTION_ID</code>	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
<code>RECORD_DOES_NOT_EXIST</code>	The specified record instance does not exist.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
var result = actionMod.execute({id: 'note', recordType: 'timebill', params: {recordId:1}
});
...
// Add additional code
```

action.execute.promise(options)

Method Description	Executes the record action asynchronously. If the action fails, it is listed in the results object's <code>notifications</code> property. If the action executes successfully, the <code>notifications</code> property is usually empty.
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/action Module
Sibling Object Members	Action Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.recordType	string	required	The record type. For a list of record types, see record.Type .
options.id	string	required	The action ID. For a list of action IDs, see Supported Record Actions .
options.params	Object	required	Action arguments.
options.params.recordId	string	required	The record instance ID. This is the NetSuite record internal ID.

Errors

Error Code	Thrown If
SSS_INVALID_RECORD_TYPE	The specified record type is invalid.
SSS_MISSING_REQD_ARGUMENT	A required parameter is missing.
SSS_INVALID_ACTION_ID	The specified action does not exist on the specified record type. – or – The action exists, but cannot be executed on the specified record instance.
RECORD_DOES_NOT_EXIST	The specified record instance does not exist.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/action Module Script Sample](#).

```
// Add additional code
...
actionMod.execute.promise({id: 'note', recordType: 'timebill', params: {recordId: 1}}).then(function(result) {
    // do something with the result
});
...
// Add additional code
```

N/auth Module

Load the N/auth module when you want to change your NetSuite login credentials.

- [N/auth Module Members](#)
- [N/auth Module Script Sample](#)

N/auth Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	auth.changeEmail(options)	void	Server-side scripts	Changes the current user's NetSuite email address (user name).
	auth.changePassword(options)	void	Server-side scripts	Changes the current user's NetSuite password.

N/auth Module Script Sample

The following example changes the currently logged-in user's NetSuite email address and password.

This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record).



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

For help with scripting in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 */
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();
});
```

auth.changeEmail(options)

Method Description	Method used to change the current user's NetSuite email address (user name).
Returns	void
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units

Module	N/auth Module
Since	Version 2015 Release 2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.password</code>	string	required	<ul style="list-style-type: none"> The logged in user's NetSuite password. 	Version 2015 Release 2
<code>options.newEmail</code>	string	required	<ul style="list-style-type: none"> The logged in user's NetSuite email address. 	Version 2015 Release 2
<code>options.onlyThisAccount</code>	boolean true false	optional	<ul style="list-style-type: none"> If set to <code>true</code>, the email address change is applied only to roles within the current account. If set to <code>false</code>, the email address change is applied to all accounts and roles. The default value is <code>true</code>. 	Version 2015 Release 2

Errors

Error Code	Message	Thrown If
INVALID_PSWD		The argument for <code>options.password</code> is invalid.
INVALID_EMAIL		The argument for <code>options.newEmail</code> is invalid.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/auth Module Script Sample](#).

```
//Add additional code
...
auth.changeEmail({
  password: 'mypwd',
  newEmail: 'jwolf@netsuite.com',
  onlyThisAccount: true
});
...
//Add additional code
```

auth.changePassword(options)

Method Description	Method used to change the current user's NetSuite password.
Returns	void

Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/auth Module
Since	Version 2015 Release 2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.currentPassword</code>	string	required	<ul style="list-style-type: none"> The logged in user's current NetSuite password. 	Version 2015 Release 2
<code>options.newPassword</code>	string	required	<ul style="list-style-type: none"> The logged in user's new NetSuite password. 	Version 2015 Release 2

Errors

Error Code	Message	Thrown If
INVALID_PSWD		The argument for <code>options.currentPassword</code> is invalid.
USER_ERROR		

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/auth Module Script Sample](#).

```
//Add additional code
...
auth.changePassword({
  currentPassword: 'mycurrentPWD',
  newPassword: 'mynewPWD'
});
...
//Add additional code
```

N/cache Module

Load the cache module to enable temporary, short-term storage of data. Data is stored in the cache according to its specified time to live, or ttl. The ttl is specified in the [Cache.put\(options\)](#) method `options.ttl` parameter. The cache module is supported by all server-side script types.

Using a cache improves performance by eliminating the need for scripts in your account to retrieve the same piece of data more than one time. You can create a cache that is accessible at any of three levels:

A cache can be available to the current script only, to all server-side scripts in the current bundle, or to all server-side scripts in your NetSuite account.

- [N/cache Module Members](#)
- [Cache Object Members](#)
- [N/cache Module Script Sample](#)

N/cache Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	cache.Cache	Object	Server-side scripts	Encapsulates a segment of memory that can be used to temporarily store data on a short-term basis.
Method	cache.getCache(options)	cache.Cache	Server-side scripts	Checks for a cache object with the specified name. If the cache exists, this method returns the cache object. If the cache does not exist, the system creates it.
Enum	cache.Scope	enum	Server-side scripts	An enum used to populate the <code>Cache.scope</code> property.

Cache Object Members

The following members are called on [cache.Cache](#).

Member Type	Name	Return Type/ Value Type	Supported Script Types	Description
Method	Cache.get(options)	string	Server-side scripts	Retrieves a value from the cache based on a key that you provide. If the requested value is not present in the cache, the method calls the user-defined function identified by the method's <code>option.loader</code> parameter. If the value provided by that function is not a string, the system uses <code>JSON.stringify()</code> to convert it. The string value is then cached and returned.
	Cache.put(options)	string	Server-side scripts	Puts a value into the cache. If the value provided is not a string, the system uses <code>JSON.stringify()</code> to convert the value to a string.
	Cache.remove(options)	string	Server-side scripts	Removes a value from the cache.
Property	Cache.name	string	Server-side scripts	A label that identifies the cache.
	Cache.scope	string	Server-side scripts	A value that describes the availability of the cache. A cache can be made available to the current script only, to all scripts in the current bundle, or to all scripts in your NetSuite account.

N/cache Module Script Sample

The following sample Suitelet retrieves the name of a city based on a ZIP code. To speed processing, the Suitelet uses a cache.

In this sample, ZIP code is the key used to retrieve city names from the cache. For any ZIP code provided, if the corresponding city value is not already stored in the cache, a loader function is called. This function, called `zipCodeDatabaseLoader`, loads a CSV file and uses it to find the requested value. (The `zipCodeDatabaseLoader` is shown in the next script sample.)

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

For help with scripting in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Note: This sample script uses the `define` function. Note that you cannot use [On Demand Debugging](#) to step through a `define` function. You must use [Deployed Debugging](#) to step through this script.

```
/**
 * @NScriptType Suitelet
 * @NApiVersion 2.x
 */
define(['N/cache', '/SuiteScripts/zipCodes/ca/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(){
      var zipCache = cache.getCache({name: ZIP_CODES_CACHE_NAME});
      var 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){
      var start = new Date();
      if (context.request.parameters.purgeZipCache === 'true'){
        var zipCache = cache.getCache({name: ZIP_CODES_CACHE_NAME});
        zipCache.remove({key: ZIP_TO_CITY_IDX_JSON});
      }
      var 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 returns the loader function used in the preceding code sample. The loader function shows how to use a CSV file to retrieve a value that was missing from a cache. This script does not need to include logic for placing the retrieved value into the cache — whenever a value is returned through the `options.loader` parameter, the value is automatically placed into the cache. For this reason, a loader function can serve as the sole method of populating a cache with values.

```

/**
 * @NApiVersion 2.0
 * @NModuleScope Public
 */
define(['N/file', 'N/cache'], function(file, cache){
    const ZIP_CODES_CSV_PATH = 'Resources/free-zipcode-ca-database-primary.csv';
    function trimOuterQuotes(str){
        return (str || '').replace(/^"+/, '').replace(/"+$/, '');
    }

    function zipCodeDatabaseLoader(context){
        log.audit('Loading Zip Codes for ZIP_CODES_CACHE');
        var zipCodesCsvText = file.load({id: ZIP_CODES_CSV_PATH}).getContents();
        var zipToCityIndex = {};
        var 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
    }
});

```

cache.Cache

Object Description	A segment of memory that can be used to temporarily store data needed by one script, by all scripts in a bundle, or by all scripts in the NetSuite account. This object is returned by cache.getCache(options) .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/cache Module
Methods and Properties	Cache Object Members

Since	2016.2
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myCache = cache.getCache({
    name: 'temporaryCache',
    scope: cache.Scope.PRIVATE
});
...
//Add additional code
```

Cache.get(options)

Method Description	This method retrieves a string value from the cache. The value retrieved is identified by a key that you pass by using the options.key parameter. If a requested value is not present in the cache, the system calls the function identified by the options.loader parameter. This user-defined function should provide some logic for retrieving a value that is not in the cache.
Returns	String or null
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	1 unit if the value is present in the cache; 2 units if the loader function is used
Module	N/cache Module
Parent Object	cache.Cache
Sibling Object Members	Cache Object Members
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.key	string	required	A string that identifies the value to be retrieved from the cache. This value cannot be null.
options.loader	function	optional, but strongly recommended	A user-defined function that returns the requested value if it is not already present in the cache. Additionally, when the loader retrieves a value, the system automatically places that value in the cache. For this reason, NetSuite recommends using the loader function as the primary means of populating the cache. Note also that if the value returned by the loader is not a string, the system uses <code>JSON.stringify()</code> to convert the value before it is

Parameter	Type	Required / Optional	Description
			placed in the cache and returned. The maximum size of a value that can be placed in the cache is 500KB. When no loader is specified and a value is missing from the cache, the system returns null.
options.ttl	number	optional	<p>The maximum duration, in seconds, that a value retrieved by the loader can remain in the cache. Note that the value may be removed before the <code>ttl</code> limit is reached. The default <code>ttl</code> value is no limit. The minimum value is 300 (five minutes).</p> <div>  Important: A cached value is not guaranteed to stay in the cache for the full duration of the <code>ttl</code> value. The <code>ttl</code> value represents the maximum time that the cached value may be stored. </div>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myValue = myCache.get({
    key: 'keyText',
    loader: loader,
    ttl: 18000
});
...
//Add additional code
```

Cache.put(options)

Method Description	<p>Use this method to place a value into a cache.</p> <div>  Note: You can also place a value in a cache by using the Cache.get(options) method. In general, using the get method may result in a more efficient design. </div>
Returns	Void
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	1 unit
Module	N/cache Module
Parent Object	cache.Cache
Sibling Object Members	Cache Object Members
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.key	string	required	An identifier of the value that is being cached. The maximum size of the cache key is 4 kilobytes.
options.value	string	required	The value to place in the cache. If the value submitted is not a string, the system uses <code>JSON.stringify()</code> to convert the value before it is placed in the cache. The maximum size of the value is 500KB.
options.ttl	number	optional	<p>The maximum duration, in seconds, that the value may remain in the cache. Note that the value may be removed before the <code>ttl</code> limit is reached. The default <code>ttl</code> value is no limit. The minimum value is 300 (five minutes).</p> <div>  Important: A cached value is not guaranteed to stay in the cache for the full duration of the <code>ttl</code> value. The <code>ttl</code> value represents the maximum time that the cached value may be stored. </div>

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
myCache.put({
  key: 'keyText',
  value: 'valueText',
  ttl: 300
});
...
//Add additional code
```

Cache.remove(options)

Method Description	Removes a value from the cache.
Returns	Void
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	1 unit
Module	N/cache Module
Parent Object	cache.Cache
Sibling Object Members	Cache Object Members

Since	2016.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.key</code>	string	required	An identifier of the value that is being removed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
myCache.remove({
  key: 'keyText'
});
...
//Add additional code
```

Cache.name

Property Description	A label that identifies a cache.
Type	string
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/cache Module
Parent Object	cache.Cache
Sibling Object Members	Cache Object Members
Since	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myCache = cache.getCache({
  name: 'temporaryCache',
  scope: cache.Scope.PRIVATE
});
...
//Add additional code
```

Cache.scope

Property Description	A label that describes the availability of the cache to other scripts.
Type	cache.Scope
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/cache Module
Parent Object	cache.Cache
Sibling Object Members	Cache Object Members
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myCache = cache.getCache({
    name: 'temporaryCache',
    scope: cache.Scope.PRIVATE
});
...
//Add additional code
```

cache.getCache(options)

Method Description	Method used to create a new cache.Cache object.
Returns	cache.Cache
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	n/a
Module	N/cache Module
Sibling Module Members	N/cache Module Members
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	required	A label that will identify the cache you are creating. The maximum size of the cache name is 1 kilobyte.

Parameter	Type	Required / Optional	Description
options.scope	string	optional, but if you do not set a value, the default of PRIVATE is used	This value is set with the cache.Scope enum. It determines the availability of the cache. A cache can be made available to the current script only, to all scripts in the current bundle, or to all scripts in your NetSuite account.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myCache = cache.getCache({
    name: 'temporaryCache',
    scope: cache.Scope.PRIVATE
});
...
//Add additional code
```

cache.Scope

Enum Description	Enumeration that holds string values that describe the availability of the cache. This enum is used to set the value of the Cache.scope property.
	<div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/cache Module
Sibling Module Members	N/cache Module Members
Since	2016.2

Values

Value	Description
PRIVATE	The cache is available only to the current script. This value is the default.
PROTECTED	The cache is available only to some scripts, as follows: <ul style="list-style-type: none"> ■ If the script is part of a bundle, the cache is available to all scripts in the same bundle. ■ If the script is not in a bundle, then the cache is available to all scripts not in a bundle.

Value	Description
PUBLIC	The cache is available to any script in the NetSuite account.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/cache Module Script Sample](#).

```
//Add additional code
...
var myCache = cache.getCache({
    name: 'temporaryCache',
    scope: cache.Scope.PRIVATE
});
...
//Add additional code
```

N/config Module

Load the N/config module when you want to access NetSuite configuration settings. The [config.load\(options\)](#) method returns a [record.Record](#) object. Use the [record.Record](#) object members to access configuration settings. You do not need to load the record module to do this.

See [config.Type](#) for a list of supported configuration objects.

- [N/config Module Members](#)
- [N/config Module Script Sample](#)

N/config Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	config.load(options)	record.Record	Server-side scripts	Loads a record.Record object that encapsulates the specified configuration page.
Enum	config.Type	enum	Server-side scripts	Holds the string values for supported configuration objects. This enum is used to set the value of the NetSuite configuration page you want to access.

N/config Module Script Sample



Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

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

For help with scripting in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 *@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();
});
```

config.load(options)

Method Description	<p>Method used to load a record.Record object that encapsulates the specified NetSuite configuration page.</p> <p>After the configuration page loads, all preference names and IDs are available to get or set. For more information, see the help topic Preference Names and IDs.</p> <p>You can use the following Record Object Members to get and set preference names and IDs:</p> <ul style="list-style-type: none"> ■ Record.getField(options) ■ Record.getFields() ■ Record.getText(options) ■ Record.getValue(options) ■ Record.setText(options) ■ Record.setValue(options)
Returns	record.Record
Supported Script Types	<p>Server-side scripts</p> <p>For additional information, see SuiteScript 2.0 Script Types.</p>

Governance	10 usage units
Module	N/config Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	enum	required	The NetSuite configuration page you want to access. Use the config.Type enum to set the value.	2015.2
options.isDynamic	boolean true false	optional	<p>Determines whether the record is loaded in dynamic mode.</p> <ul style="list-style-type: none"> ■ If set to <code>true</code>, the record is loaded in dynamic mode. ■ If set to <code>false</code>, the record is loaded in standard mode. <p>For more information, see SuiteScript 2.0 – Standard and Dynamic Modes.</p>	2015.2

Error Code	Message	Thrown If
INVALID_RCRD_TYPE	The record type {type} is invalid.	The <code>type</code> argument is invalid or missing.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/config Module Script Sample](#).

```
//Add additional code
...
var configRecObj = config.load({
    type: config.Type.COMPANY_INFORMATION
});
configRecObj.setText({
    fieldId: 'fiscalmonth',
    text: 'July'
});
configRecObj.save();
...
//Add additional code
```

config.Type

Enum Description	Enumeration that holds the string values for supported configuration pages.
-------------------------	---

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Module	N/config Module
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Since	2015.2

Values

Value	Configuration Page
USER_PREFERENCES	Set Preferences page (Home > Set Preferences) For more information about the fields on the page, see the help topic User Preferences .
COMPANY_INFORMATION	Company Information page (Setup > Company > Company Information) For more information about the fields on the page, see the help topic Company Information .
COMPANY_PREFERENCES	General Preferences page (Setup > Company > General Preferences) For more information about the fields on the page, see the help topic General Preferences .
ACCOUNTING_PREFERENCES	Accounting Preferences page (Setup > Accounting > Accounting Preferences) For more information about the fields on the page, see the help topic Accounting Preferences .
ACCOUNTING_PERIODS	Accounting Periods page (Setup > Accounting > Manage Accounting Periods) For more information about the fields on the page, see the help topic Accounting Periods .
TAX_PERIODS	Tax Periods page (Setup > Accounting > Manage Tax Periods) For more information about the fields on the page, see the help topic Tax Periods .
FEATURES	Enable Features page (Setup > Company > Enable Features) For more information about feature names and IDs, see the help topic Feature Names and IDs .
TIME_POST	For additional information, see the help topic Posting Time Transactions .
TIME_VOID	For additional information, see the help topic Posting Time Transactions .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/config Module Script Sample](#).

```
//Add additional code
...
```

```

var configRecObj = config.load({
    type: config.Type.COMPANY_INFORMATION
});
configRecObj.setText({
    fieldId: 'fiscalmonth',
    text: 'July'
});
configRecObj.save();
...
//Add additional code

```

N/crypto Module

The N/crypto module encapsulates hashing, hash-based message authentication (hmac), and symmetrical encryption.

When the crypto module is used, SuiteScript also loads [N/encode Module](#).

- [N/crypto Module Members](#)
- [Cipher Object Members](#)
- [CipherPayload Object Members](#)
- [Decipher Object Members](#)
- [Hash Object Members](#)
- [Hmac Object Members](#)
- [SecretKey Object Members](#)
- [N/crypto Module Script Samples](#)

N/crypto Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	crypto.Cipher	Object	Server-side scripts	Encapsulates a cipher.
	crypto.CipherPayload	Object	Server-side scripts	Encapsulates a cipher payload.
	crypto.Decipher	Object	Server-side scripts	Encapsulates a decipher.
	crypto.Hash	Object	Server-side scripts	Encapsulates a hash.
	crypto.Hmac	Object	Server-side scripts	Encapsulates an hmac.
	crypto.SecretKey	Object	Server-side scripts	Encapsulates a secret key handle.
Method	crypto.createCipher(options)	Object	Server-side scripts	Creates and returns a new crypto.Cipher Object.
	crypto.createDecipher(options)	Object	Server-side scripts	Creates and returns a new crypto.Decipher object.
	crypto.createHash(options)	Object	Server-side scripts	Creates and returns a new crypto.Hash Object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>crypto.createHmac(options)</code>	Object	Server-side scripts	Creates and returns a new <code>crypto.Hmac</code> Object.
	<code>crypto.createSecretKey(options)</code>	Object	Server-side scripts	Creates and returns a new <code>crypto.SecretKey</code> Object.
Enum	<code>crypto.EncryptionAlg</code>	string (read-only)	Server-side scripts	Holds the string values for supported encryption algorithms. Sets the <code>options.algorithm</code> parameter for <code>crypto.createCipher(options)</code> .
	<code>crypto.HashAlg</code>	string (read-only)	Server-side scripts	Holds the string values for supported hashing algorithms. Sets the value of the <code>options.algorithm</code> parameter for <code>crypto.createHash(options)</code> and <code>crypto.createHmac(options)</code> .
	<code>crypto.Padding</code>	string (read-only)	Server-side scripts	Holds the string values for supported cipher padding. Sets the <code>options.padding</code> parameter for <code>crypto.createCipher(options)</code> and <code>crypto.createDecipher(options)</code> .

Cipher Object Members

The following members are called on `crypto.Cipher`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Cipher.update(options)</code>	Object	Server-side scripts	Updates the clear data with the specified encoding
	<code>Cipher.final(options)</code>	Object	Server-side scripts	Returns the cipher data.

CipherPayload Object Members

The following members are called on `crypto.CipherPayload`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>CipherPayload.ciphertext</code>	string	Server-side scripts	The result of the ciphering process.
	<code>CipherPayload.iv</code>	number	Server-side scripts	An initialization vector

Decipher Object Members

The following members are called on `crypto.Decipher`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Decipher.final(options)</code>	string	Server-side scripts	Returns the clear data.
	<code>Decipher.update(options)</code>	void	Server-side scripts	Updates cipher data with the specified encoding.

Hash Object Members

The following members are called on [crypto.Hash](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Hash.digest(options)	string	Server-side scripts	Calculates the digest of the data to be hashed.
	Hash.update(options)	void	Server-side scripts	Updates the clear data with the encoding specified.

Hmac Object Members

The following members are called on [crypto.Hmac](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Hmac.digest(options)	string	Server-side scripts	Gets the computed digest.
	Hmac.update(options)	void	Server-side scripts	Updates the clear data with the encoding specified.

SecretKey Object Members

The following members are called on [crypto.SecretKey](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Secretkey.guid	string	Server-side scripts	The GUID associated with the secret key.
	SecretKey.encoding	string	Server-side scripts	The encoding used for the clear text value of the secret key.

N/crypto Module Script Samples

Example 1

The following example demonstrates the APIs needed to generate a secure key using the SHA512 hashing algorithm. It is not a functional example that will work in the debugger (because the GUID does not exist in your account). Refer to the Suitelet example for a more complete usage. See [Example 2](#).

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



Note: The GUID in this sample is a placeholder. You must replace it with a valid value from your NetSuite account.

For help with scripting in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
```

```

* @NApiVersion 2.x
*/
require(['N/crypto', 'N/encode', 'N/runtime'],
function(crypto, encode, runtime) {
  function createSecureKeyWithHash() {
    var inputString = 'YWJjZGVmZwo=';
    var myGuid = '{284CFB2D225B1D76FB94D150207E49DF}';
    var sKey = crypto.createSecretKey({
      guid: myGuid,
      encoding: encode.Encoding.UTF_8
    });
    var hmacSHA512 = crypto.createHmac({
      algorithm: crypto.HashAlg.SHA512,
      key: sKey
    });
    hmacSHA512.update({
      input: inputString,
      inputEncoding: encode.Encoding.BASE_64
    });
    var digestSHA512 = hmacSHA512.digest({
      outputEncoding: encode.Encoding.HEX
    });
  }
  createSecureKeyWithHash();
});

```

Example 2



Note: This sample script uses the `define` function. Note that you cannot use [On Demand Debugging](#) to step through a `define` function. You must use [Deployed Debugging](#) to step through this script.



Important: 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'],
function(ui, runtime, crypto, encode) {
  function onRequest(option) {
    if (option.request.method === 'GET') {
      var form = ui.createForm({
        title: 'My Credential Form'
      });
      var skField = form.addSecretKeyField({
        id: 'mycredential',
        label: 'Credential',
        restrictToScriptIds: [runtime.getCurrentScript().id],
        restrictToCurrentUser: false
      });
      skField.maxLength = 200;
      form.addSubmitButton();
    }
  }
});

```

```

        option.response.writePage(form);
    } else {
        var form = ui.createForm({
            title: 'My Credential Form'
        });
        var inputString = "YWJjZGVmZwo=";
        var myGuid = option.request.parameters.mycredential;
        // Create the key
        var sKey = crypto.createSecretKey({
            guid: myGuid,
            encoding: encode.Encoding.UTF_8
        });
        try {
            var hmacSha512 = crypto.createHmac({
                algorithm: 'SHA512',
                key: sKey
            });
            hmacSha512.update({
                input: inputString,
                inputEncoding: encode.Encoding.BASE_64
            });
            var 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
};
});

```

crypto.Cipher

Object Description	Encapsulates a cipher. For a complete list of this object's methods and properties, see Cipher Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var cipher = crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey
});
...
//Add additional code
```

Cipher.final(options)

Method Description	Method used to return the cipher data. Sets the output encoding for the crypto.CipherPayload object.
Returns	A crypto.CipherPayload Object
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.outputEncoding</code>	enum	optional	The output encoding for a crypto.CipherPayload object. The default value is HEX. Use the encode.Encoding enum to set the value.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey
});

var cipherPayload = cipher.final({
  outputEncoding: encode.Encoding.BASE_64
});
```

```
...
//Add additional code
```

Cipher.update(options)

Method Description	Method used to update the clear data with the specified encoding.
Returns	Void
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.input	string	required	The clear data to be updated.
options.inputEncoding	enum	optional	The input encoding. Use the encode.Encoding enum to set the value. The default value is <code>UTF_8</code> .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var reencoded = Cipher.update({
  input: 'Carrot cake gummi bears'
});
...
//Add additional code
```

crypto.CipherPayload

Object Description	Encapsulates a cipher payload. For a complete list of this object's methods and properties, see CipherPayload Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey
});

var cipherPayload = cipher.final({
  outputEncoding: encode.Encoding.HEX
});
...
//Add additional code
```

CipherPayload.ciphertext

Property Description	The result of the ciphering process. For example, to take the cipher payload and send it to another system.
Type	string
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
log.debug({
  title: "Ciphertext: ",
  details: cipherPayload.ciphertext
});
...
//Add additional code
```

CipherPayload.iv

Property Description	Initialization vector for the cipher payload. You can pass in the iv value to crypto.createDecipher(options)
Type	string
Supported Script Types	Server-side scripts

	For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
log.debug({
  title: "CipherPayload IV: ",
  details: cipherPayload.iv
});
...
//Add additional code
```

crypto.Decipher

Object Description	Encapsulates a decipher. This object has methods that decrypt. For a complete list of this object's methods and properties, see Decipher Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
crypto.createDecipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey
});
...
//Add additional code
```

Decipher.final(options)

Method Description	Method used to return the clear data.
Returns	string
Supported Script Types	Server-side scripts

	For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.outputEncoding	string	optional	Specifies the encoding for the output Set the value using the encode.Encoding enum. The default value is <code>UTF_8</code> .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var decipher1 = Decipher.final({
    outputEncoding: encode.Encoding.HEX
});
...
//Add additional code
```

Decipher.update(options)

Method Description	Method used to update cipher data with the specified encoding.
Returns	Void
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.input	string	required	The data to update
options.inputEncoding	string	optional	Specifies the encoding of the input data

Parameter	Type	Required / Optional	Description
			Set the value using the encode.Encoding enum. The default value is HEX.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var decipher1 = Decipher.update({
  input: '73616d706c65737472696e67',
  inputEncoding: encode.Encoding.HEX
});
...
//Add additional code
```

crypto.Hash

Object Description	Encapsulates a hash. For a complete list of this object's methods and properties, see Hash Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var hashObj = crypto.createHash({
  algorithm: crypto.HashAlgorithm.SHA256
});
...
//Add additional code
```

Hash.digest(options)

Method Description	Calculates the digest of the data to be hashed.
Returns	A hash value as a string
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.outputEncoding	string	optional	The output encoding. Set using the encode.Encoding enum. The default value is <code>HEX</code> .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var digestSample = hashObj.digest({
    outputEncoding: encode.Encoding.HEX
});
...
//Add additional code
```

Hash.update(options)

Method Description	Method used to update clear data with the encoding specified.
Returns	Void
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.input	string	required	The data to be updated.
options.inputEncoding	string	optional	The input encoding. Set using the encode.Encoding enum.

Parameter	Type	Required / Optional	Description
			The default value is <code>UTF_8</code> .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var inputString = 'Lemon drops ice cream jelly marzipan cake';
hashSample.update({
  input: inputString
});
...
//Add additional code
```

crypto.Hmac

Object Description	Encapsulates an hmac. For a complete list of this object's methods and properties, see Hmac Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var hmacSHA512 = crypto.createHmac({
  algorithm: crypto.HashAlg.SHA512,
  key: sKey
});
...
//Add additional code
```

Hmac.digest(options)

Method Description	Gets the computed digest.
Returns	An hmac value as a string
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.outputEncoding	string	optional	Specifies the encoding of the output string. Set using the encode.Encoding enum. The default value is HEX.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var digestSHA512 = hmacSHA512.digest({
    outputEncoding: encode.Encoding.HEX
});
...
//Add additional code
```

Hmac.update(options)

Method Description	Method used to update the clear data with the encoding specified.
Returns	Void
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.input	string	required	The hmac data to be updated.
options.inputEncoding	enum	optional	The input encoding. Set using the encode.Encoding enum. The default value is UTF_8.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
hmacSHA512.update({
    input: inputString,
    inputEncoding: encode.Encoding.BASE_64
});
...
//Add additional code
```

crypto.SecretKey

Object Description	Encapsulates the handle to the key. The handler does not store the key value. It points to the key stored within the NetSuite system. The GUID is also required to find the key. For a complete list of this object's methods and properties, see SecretKey Object Members .
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var sKey = crypto.createSecretKey({
    guid: '284CFB2D225B1D76FB94D150207E49DF',
    encoding: encode.Encoding.UTF_8
});
...
//Add additional code
```

SecretKey.encoding

Property Description	The encoding used for the clear text value of the secret key.
Type	string
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
log.debug({
  title: "Secret Key Encoding: ",
  details: sKey.encoding
});
...
//Add additional code
```

Secretkey.guid

Property Description	The GUID associated with the secret key.
Type	string
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/crypto Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
log.debug({
  title: "Secret Key GUID: ",
  details: sKey.guid
});
...
//Add additional code
```

crypto.createCipher(options)

Method Description	Method used to create and return a <code>crypto.EncryptionAlg</code> object. <div> Note: The blockCipherMode is automatically set to CBC.</div>
Returns	A <code>crypto.EncryptionAlg</code> object
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.algorithm	string	required	The hash algorithm. Set the value using the crypto.EncryptionAlg enum.	2015.2
options.key	object	required	The crypto.SecretKey object. <div> Note: When using the crypto.SecretKey object for an AES algorithm, the length of the text (secret key) that is used to generate the GUID must be 16, 24, or 32 characters.</div>	2015.2
options.padding	string	optional	The padding for the cipher text. Set the value using the crypto.Padding enum. By default, the value is set to PKCS5Padding.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var cipher = crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey,
  padding: crypto.Padding.PKCS5Padding
});
...
//Add additional code
```

crypto.createDecipher(options)

Method Description	Method used to create a crypto.Decipher object. <div> Note: The blockCipherMode is automatically set to CBC.</div>
Returns	A crypto.Decipher object.
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None

Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.algorithm	string	required	The hash algorithm. Set by the crypto.EncryptionAlg enum.	2015.2
options.key	object	required	The crypto.SecretKey object used for encryption.  Note: When using the crypto.SecretKey object for an AES algorithm, the length of the text (secret key) that is used to generate the GUID must be 16, 24, or 32 characters.	2015.2
options.padding	object	optional	The padding for the cipher. Set the value using the crypto.Padding enum.	2015.2
options.iv	string	required	The initialization vector that was used for encryption.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var decipher = crypto.createDecipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey,
  padding: NoPadding,
  iv: '2311141720'
});
...
//Add additional code
```

crypto.createHash(options)

Method Description	Method used to create a crypto.Hash object.
Returns	The crypto.Hash object created using this method.
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.algorithm</code>	string	required	<ul style="list-style-type: none"> The hash algorithm. Set using the <code>crypto.HashAlg</code> enum. 	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var hashObj = crypto.createHash({
  algorithm: crypto.HashAlg.SHA256
});
...
//Add additional code
```

crypto.createHmac(options)

Method Description	Method used to create a <code>crypto.Hmac</code> object.
Returns	A <code>crypto.Hmac</code> object.
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.algorithm</code>	string	required	The hash algorithm. Use the <code>crypto.HashAlg</code> enum to set this value.	2015.2
<code>options.key</code>	object	required	The <code>crypto.SecretKey</code> object.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
```

```

...
var hmacObj = crypto.createHmac({
  algorithm: HashAlg.SHA256,
  key: sKey
});
...
//Add additional code

```

crypto.createSecretKey(options)

Method Description	<p>Method used to create a new crypto.SecretKey object. This method can take a GUID. Use Form.addCredentialField(options) to generate a value.</p> <div>  Note: When using the crypto.SecretKey object for an AES algorithm, the length of the text (secret key) that is used to generate the GUID must be 16, 24, or 32 characters. </div>
Returns	A crypto.SecretKey object
Supported Script Types	Server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/crypto Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.guid</code>	string	required	A GUID used to generate a secret key. The GUID can resolve to either data or metadata.	2015.2
<code>options.encoding</code>	enum	optional	Specifies the encoding for the SecureKey. Set this value using the encode.Encoding enum. The default value is HEX.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```

//Add additional code
...
var secretKey = crypto.createSecretKey({
  encoding: encode.Encoding.HEX,
  guid: '284CFB2D225B1D76FB94D150207E49DF'
});
...
//Add additional code

```

crypto.EncryptionAlg

Enum Description	Holds the string values for supported encryption algorithms. Sets the <code>options.algorithm</code> parameter for <code>crypto.createCipher(options)</code> . <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Module	N/crypto Module
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Since	2015.2

Values

<div> <div></div> <div>AES</div> </div>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var cipher = crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
  key: sKey
});
...
//Add additional code
```

crypto.HashAlg

Enum Description	Holds the string values for supported hashing algorithms. Sets the value of the <code>options.algorithm</code> parameter for <code>crypto.createHash(options)</code> and <code>crypto.createHmac(options)</code> . <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Module	N/crypto Module
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Since	2015.2

Values

- SHA1
- SHA256
- SHA512
- MD5

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var hmacSHA512 = crypto.createHmac({
  algorithm: crypto.HashAlg.SHA512,
  key: sKey
});
...
//Add additional code
```

crypto.Padding

Enum Description	Holds the string values for supported cipher padding. Sets the <code>options.padding</code> parameter for <code>crypto.createCipher(options)</code> and <code>crypto.createDecipher(options)</code> . <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Module	N/crypto Module
Supported Script Types	All server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Since	2015.2

Values

- NoPadding
- PKCS5Padding

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var cipher = crypto.createCipher({
  algorithm: crypto.EncryptionAlg.AES,
```

```

    key: sKey,
    padding: crypto.Padding.NoPadding
  });
  ...
  //Add additional code

```

N/currency Module

Load the N/currency module when you want to work with exchange rates within your NetSuite account. You can use this module to find the exchange rate between two currencies based on a certain date.

To use multiple currencies, the Multiple Currencies feature must be enabled. For information on enabling this feature, see the help topic [Enabling the Multiple Currencies Feature](#).

Note: Currency formatting is handled by the [N/format Module](#).

- [N/currency Module Member](#)
- [N/currency Module Script Sample](#)

N/currency Module Member

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	currency.exchangeRate(options)	number	Client and server-side scripts	Returns an exchange rate between two currencies.

N/currency Module Script Sample

The following example obtains the exchange rate between the Canadian dollar and the US dollar on July 28, 2015.

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```

/**
 *@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();
  }

```


});

currency.exchangeRate(options)

Method Description	<p>Method used to return the exchange rate between two currencies based on a certain date. The source currency is looked up relative to the target currency on the effective date. For example, if use British pounds for the source and US dollars for the target and the method returns '1.52', this means that if you were to enter an invoice today for a GBP customer in your USD subsidiary, the rate would be 1.52.</p> <p>The exchange rate values are sourced from the Currency Exchange Rate record.</p> <div>  Note: The Currency Exchange Rate record itself is not a scriptable record. </div>
Returns	The exchange rate as a decimal number
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/currency Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.date	Date	optional	<ul style="list-style-type: none"> Pass in a new Date object. For example, date: new Date('7/28/2015') If date is not specified, then it defaults to today (the current date). The date determines the exchange rate in effect. If there are multiple rates, it is the latest entry on that date. Use the same date format as your NetSuite account. 	2015.2
options.source	number string	required	<ul style="list-style-type: none"> The internal ID or three-letter ISO code for the currency you are converting from. For example, you can use either 1 (internal ID) or USD (currency code). If the Multiple Currencies feature is enabled, from your account, you can view a list of all the currency internal IDs and ISO codes at Lists > Accounting > Currencies. 	2015.2
options.target	number string	required	<ul style="list-style-type: none"> The internal ID or three-letter ISO code for the currency you are converting to. 	2015.2

Errors

Error Code	Message	Thrown If
MISSING_REQD_ARGUMENT	exchangeRate: Missing a required argument: <source/target>	The source or target argument is missing.

Error Code	Message	Thrown If
SSS_INVALID_CURRENCY_ID	You have entered an invalid currency symbol or internal ID: <target/source>	The <code>source</code> or <code>target</code> argument is invalid. If the Multiple Currencies feature is enabled, from your account, you can view a list of currency internal IDs and ISO codes at Lists > Accounting > Currencies.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currency Module Script Sample](#).

```
//Add additional code
...
var canadianAmount = 100;
var rate = currency.exchangeRate({
    source: 'CAD',
    target: 'USD',
    date: new Date('7/28/2015')
});
var usdAmount = canadianAmount * rate;
...
//Add additional code
```

N/currentRecord Module

You use the N/currentRecord module to access the record that is active in the current client-side context. Be aware that when the current record is in view mode it cannot be edited; it is a read-only record when in view mode. As such, any set APIs do not work on the current record in view mode.

You can use the currentRecord module in the following types of scripts:

- **Entry point client scripts** — These scripts use the `@NScriptType ClientScript` annotation. (For details, see [SuiteScript 2.0 JSDoc Validation](#).) The system automatically provides this type of script with a `currentRecord.CurrentRecord` object that represents the current record. For this reason, an entry point client script does not have to explicitly load the currentRecord module. To access the currentRecord object, create a variable and initialize it to the value of the `scriptContext.currentRecord` property, which is available in each of the [SuiteScript 2.0 Client Script Entry Points and API](#). For an example, see [SuiteScript Client Script Sample](#).
- **Client-side custom modules** — These scripts do not use an `@NScriptType` annotation (see [SuiteScript 2.0 Custom Modules](#)). For these scripts, you must manually load the currentRecord module by naming it in the script's define statement. Additionally, you must actively retrieve a `currentRecord.CurrentRecord` object by using the `currentRecord.get()` or `currentRecord.get.promise()` method. For an example, see [N/currentRecord Module Script Sample](#).

Like the [N/record Module](#), the currentRecord module provides access to body and sublist fields. However, the record module is recommended for server scripts and for cases where a client-side script needs to interact with a record other than the currently active record. By contrast, the currentRecord module is recommended for client-side scripts that need to interact with the currently active record.

Additionally, the functionality of the two modules varies slightly. For example, the currentRecord module does not permit the editing of subrecords, although subrecords can be retrieved in view mode. For additional details, see the following topics:

- [N/currentRecord Module Members](#)
- [Column Object Members](#)
- [CurrentRecord Object Members](#)
- [Field Object Members](#)
- [Sublist Object Members](#)
- [N/currentRecord Module Script Sample](#)

Note: SuiteScript supports working with standard NetSuite records and with instances of custom record types. Supported standard record types are described in the [SuiteScript Records Browser](#). Refer also to [SuiteScript Supported Records](#). For help interacting with an instance of a custom record type, see the help topic [Custom Record](#).

N/currentRecord Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	currentRecord.Column	Object	Client scripts	Encapsulates a column of a sublist on the current record.
	currentRecord.CurrentRecord	Object	Client scripts	Represents the record active on the current page.
	currentRecord.Field	Object	Client scripts	Represents a body or sublist field.
	currentRecord.Sublist	Object	Client scripts	Represents a sublist.
Method	currentRecord.get()	currentRecord.CurrentRecord	Client scripts	Retrieves a record object that represents the current record.
	currentRecord.get.promise()	Promise	Client scripts	Retrieves a promise for an object that represents the current record.

Column Object Members

The following members are called on the [currentRecord.Column](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Column.id	string (read-only)	Client scripts	Returns the internal ID of the column.
	Column.label	string (read-only)	Client scripts	Returns the UI label for the column.
	Column.sublistId	string (read-only)	Client scripts	Returns the internal ID of the standard or custom sublist that contains the column.
	Column.type	string (read-only)	Client scripts	Returns the column type.

CurrentRecord Object Members

The following members are called on the `currentRecord.CurrentRecord` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>CurrentRecord.cancelLine(options)</code>	<code>currentRecord.CurrentRecord</code>	Client scripts	Cancels the changes made to the currently selected line.
	<code>CurrentRecord.commitLine(options)</code>	<code>currentRecord.CurrentRecord</code>	Client scripts	Commits the currently selected line.
	<code>CurrentRecord.findMatrixSublistLineValue(options)</code>	number	Client scripts	Returns the line number of the first line that contains the specified value in the matrix column.
	<code>CurrentRecord.findSublistLineWithNumber(options)</code>	number	Client scripts	Gets the line number for the first occurrence of a field value in a sublist.
	<code>CurrentRecord.getCurrentMatrixSublistLineValue(options)</code>	number Date string array boolean true false	Client scripts	Gets the value for the currently selected line in the matrix.
	<code>CurrentRecord.getCurrentSublistLineNumber(options)</code>	number	Client scripts	Gets the line number of the currently selected line.
	<code>CurrentRecord.getCurrentSublistSubrecord(options)</code>	<code>currentRecord.CurrentRecord</code>	Client scripts	Gets the subrecord for the associated sublist field on the current line. The subrecord object is retrieved in view mode.
	<code>CurrentRecord.getCurrentSublistTextValue(options)</code>	number Date string array boolean true false	Client scripts	Gets the value of the field in the currently selected line by text representation.
	<code>CurrentRecord.getCurrentSublistValue(options)</code>	number Date string array boolean true false	Client scripts	Gets the value of the field in the currently selected line.
	<code>CurrentRecord.getField(options)</code>	<code>currentRecord.Field</code>	Client scripts	Gets a field object from the record.
	<code>CurrentRecord.getLineCount(options)</code>	number	Client scripts	Returns the number of lines in the sublist.
	<code>CurrentRecord.getMatrixHeaderCount(options)</code>	number	Client scripts	Returns the number of columns for the specified matrix.
	<code>CurrentRecord.getMatrixHeaderField(options)</code>	<code>currentRecord.Field</code>	Client scripts	Gets the field for the specified header in the matrix.
	<code>CurrentRecord.getMatrixHeaderValue(options)</code>	number Date string array boolean true false	Client scripts	Gets the value for the associated header in the matrix.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	CurrentRecord.getMatrixSublistField(options)	currentRecord.Field	Client scripts	Gets the field for the specified sublist in the matrix.
	CurrentRecord.getMatrixSublistValue(options)	number Date string array boolean true false	Client scripts	Gets the value for the associated field in the matrix.
	CurrentRecord.getSublist(options)	currentRecord.Sublist	Client scripts	Gets the specified sublist object.
	CurrentRecord.getSublistField(options)	currentRecord.Field	Client scripts	Gets the specified field object from the sublist.
	CurrentRecord.getSublistText(options)	string	Client scripts	Gets the value of the field in a sublist by a string representation.
	CurrentRecord.getSublistValue(options)	number Date string array boolean true false	Client scripts	Gets the value of the field in a sublist.
	CurrentRecord.getSubrecord(options)	currentRecord.CurrentRecord	Client scripts	Gets the subrecord associated with the field. The subrecord object is retrieved in view mode.
	CurrentRecord.getText(options)	string	Client scripts	Gets the value of the field by a string representation.
	CurrentRecord.getValue(options)	number Date string array boolean true false	Client scripts	Gets the value of the field.
	CurrentRecord.hasCurrentSublistSubrecord(options)	boolean true false	Client scripts	Returns a value indicating whether the associated sublist field has a subrecord on the current line.
	CurrentRecord.hasSublistSubrecord(options)	boolean true false	Client scripts	Returns a value indicating whether the associated sublist field contains a subrecord.
	CurrentRecord.hasSubrecord(options)	boolean true false	Client scripts	Indicates whether the field has a subrecord.
	CurrentRecord.insertLine(options)	currentRecord.CurrentRecord	Client scripts	Inserts a new line in a sublist.
	CurrentRecord.removeCurrentSublistSubrecord(options)	currentRecord.CurrentRecord	Client scripts	Removes the subrecord for the associated sublist field on the current line.
	CurrentRecord.removeLine(options)	currentRecord.CurrentRecord	Client scripts	Removes a line from a sublist.
	CurrentRecord.removeSubrecord(options)	currentRecord.CurrentRecord	Client scripts	Removes the subrecord associated with the field.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	CurrentRecord.selectLine(options)	void	Client scripts	Selects a line item in a sublist.
	CurrentRecord.selectNewLine(options)	currentRecord.CurrentRecord	Client scripts	Selects a new line at the end of the sublist.
	CurrentRecord.setCurrentMatrixSublistValue(options)	currentRecord.CurrentRecord	Client scripts	Sets the value for the currently selected line in the matrix.
	CurrentRecord.setCurrentSublistText(options)	currentRecord.CurrentRecord	Client scripts	Sets the value of the field in the currently selected line using a string representation.
	CurrentRecord.setCurrentSublistValue(options)	currentRecord.CurrentRecord	Client scripts	Sets the value of the field in the currently selected line.
	CurrentRecord.setMatrixHeaderValue(options)	currentRecord.CurrentRecord	Client scripts	Sets the value for the associated header in the matrix.
	CurrentRecord.setMatrixSublistValue(options)	currentRecord.CurrentRecord	Client scripts	Sets the value for the associated field in the matrix.
	CurrentRecord.setText(options)	currentRecord.CurrentRecord	Client scripts	Sets the value of the field using a string representation.
	CurrentRecord.setValue(options)	currentRecord.CurrentRecord	Client scripts	Sets the value of the field.
Property	CurrentRecord.id	number (read-only)	Client scripts	Returns the internal record ID.
	CurrentRecord.isDynamic	boolean true false (read-only)	Client scripts	Indicates whether the record is dynamic.
	CurrentRecord.type	string (read-only)	Client scripts	Returns the record type.

Field Object Members

The following members are called on the [currentRecord.Field](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Field.getSelectOptions(options)	array	Client scripts	Returns an array of available options on a standard or custom select, multiselect, or radio field as key-value pairs. Only the first 1,000 available options are returned.
	Field.insertSelectOption(options)	void	Client scripts	Inserts an option into certain types of select and multiselect fields. This method is usable only in fields that were added by a front-end Suitelet or beforeLoad user event script.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Field.removeSelectOption(options)	void	Client scripts	Removes an option from certain types of select and multiselect fields. This method is usable only in fields that were added by a front-end Suitelet or beforeLoad user event script. It is supported only in client scripts..
Object	Field.id	string (read-only)	Client scripts	Returns the internal ID of a standard or custom body or sublist field.
	Field.isDisabled	boolean true false	Client scripts	Returns true if the standard or custom field is disabled on the record form, or false otherwise.
	Field.isDisplay	boolean true false	Client scripts	Returns true if the field is set to display on the record form, or false otherwise. This property is read-only for sublist fields.
	Field.isMandatory	boolean true false	Client scripts	Returns true if the standard or custom field is mandatory on the record form, or false otherwise.
	Field.isPopup	boolean true false (read-only)	Client scripts	Returns true if the field is a popup list field, or false otherwise.
	Field.isReadOnly	boolean true false	Client scripts	Returns true if the field on the record form cannot be edited, or false otherwise. For textarea fields, this property can be read or written to. For all other fields, this property is read-only.
	Field.isVisible	boolean true false (read-only)	Client scripts	Returns true if the field is visible on the record form, or false otherwise.
	Field.label	string (read-only)	Client scripts	Returns the UI label for a standard or custom field body or sublist field.
	Field.sublistId	string (read-only)	Client scripts	Returns the ID of the sublist associated with the specified sublist field.
	Field.type	string (read-only)	Client scripts	Returns the type of a body or sublist field.

Sublist Object Members

The following members are called on the [currentRecord.Sublist](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Sublist.getColumn(options)	currentRecord.Column	Client scripts	Returns a column in the sublist.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Sublist.id	string (read-only)	Client scripts	Returns the internal ID of the sublist.
	Sublist.isChanged	boolean true false (read-only)	Client scripts	Indicates whether the sublist has changed on the current record form.
	Sublist.isDisplay	boolean true false (read-only)	Client scripts	Indicates whether the sublist is displayed on the current record form.
	Sublist.type	string (read-only)	Client scripts	Returns the sublist type.

N/currentRecord Module Script Sample

The following example is a custom module client script named `clientDemo.js`. This script updates fields on the current record. After you upload `clientDemo.js` to a NetSuite account, it can be called by other scripts, as shown in the subsequent sample.

Because `clientDemo.js` is a custom module script, it must manually load the `currentRecord` module by naming it in the `define` statement. Additionally, it must actively retrieve a `CurrentRecord` object. It does so by using the `currentRecord.get()` method.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```

/**
 * @NApiVersion 2.0
 */
define(['N/currentRecord'], function(currentRecord) {
    return({
        test_set_getValue: function() {
            var record = currentRecord.get();
            record.setValue({
                fieldId: 'custpage_textfield',
                value: 'Body value',
                ignoreFieldChange: true,
                fireSlavingSync: true
            });
            var actValue = record.getValue({
                fieldId: 'custpage_textfield'
            });
            record.setValue({
                fieldId: 'custpage_resultfield',
                value: actValue,
                ignoreFieldChange: true,
                fireSlavingSync: true
            });
        },

        test_set_getCurrentSublistValue: function() {

```



```

var record = currentRecord.get();
record.setCurrentSublistValue({
    sublistId: 'itemvendor',
    fieldId: 'custpage_subtextfield',
    value: 'Sublist Value',
    ignoreFieldChange: true,
    fireSlavingSync: true
});
var actValue = record.getCurrentSublistValue({
    sublistId: 'itemvendor',
    fieldId: 'custpage_subtextfield'
});
record.setValue({
    fieldId: 'custpage_sublist_resultfield',
    value: actValue,
    ignoreFieldChange: true,
    fireSlavingSync: true
});
},
});
});

```

The following example 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. The use case for this example is to set up a page, adding fields and buttons, so that you can use the code you made in the first example, and see the fields and buttons in action.

Note: This sample script uses the `define` function. Note that you cannot use [On Demand Debugging](#) to step through a `define` function. You must use [Deployed Debugging](#) to step through this script.

```

/**
 * @NapiVersion 2.0
 * @NScriptType UserEventScript
 * @NModuleScope SameAccount
 */
define([], function() {
    return {
        beforeLoad: function (params)
        {
            var form = params.form;

            var textfield = form.addField({
                id: 'custpage_textfield',
                type: 'text',
                label: 'Text'
            });

            var resultfield = form.addField({
                id: 'custpage_resultfield',
                type: 'text',
                label: 'Result'
            });

            var sublistResultfield = form.addField({
                id: 'custpage_sublist_resultfield',

```

```

        type: 'text',
        label: 'Sublist Result Field'
    });

    var sublistObj = form.getSublist({
        id: 'itemvendor'
    });
    var subtextfield = sublistObj.addField({
        id: 'custpage_subtextfield',
        type: 'text',
        label: 'Sublist Text Field'
    });

    form.clientScriptModulePath = './clientDemo.js';
    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'
    });
    }
};
});

```

currentRecord.Column

Object Description	Encapsulates a column of a sublist on the current record. For a complete list of this object's properties, see Column Object Members .
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
var objColumn = objSublist.getColumn({
    fieldId: 'item'
});
...
//Add additional code

```

Column.id

Property Description	Returns the internal ID of the column.
Type	string (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code ...
var columnid = objColumn.id;
...
//Add additional code
```

Column.label

Property Description	Returns the internal ID of the column.
Type	string (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var columnlabel = objColumn.label;
...
//Add additional code
```

Column.sublistId

Property Description	Returns the internal ID of the standard or custom sublist that contains the column.
----------------------	---

Type	string (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var sublistid = objColumn.sublistId;
...
//Add additional code
```

Column.type

Property Description	Returns the column type.
Type	string (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var columntype = objColumn.type;
...
//Add additional code
```

currentRecord.CurrentRecord

Object Description	Encapsulates the record active on the current page.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module

Since	2016.2
-------	--------

Syntax



Important: The following code snippets show the syntax for this member. These snippets are not a functional examples. For a complete script example, see [N/currentRecord Module Script Sample](#) and [SuiteScript Client Script Sample](#).

The following snippet shows the retrieval of a currentRecord object in a custom module where the currentRecord was explicitly loaded.

```
//Add additional code
...
var objRecord = currentRecord.get();
...
//Add additional code
```

In an entry point client script, you do not have use the get method to retrieve the current record. (An entry point client script is one that uses the `@NScriptType ClientScript` annotation.) In these scripts, a currentRecord object is automatically created when the script is loaded. It is part of the context object that passed to each of the client script type's entry points. However, you do have to create a variable to represent the current record, as shown in the following snippet.

```
//Add additional code
...

function pageInit(context) {
    var currentRec = context.currentRecord;
    ...
//Add additional code
```

CurrentRecord.cancelLine(options)

Method Description	Cancels the currently selected line on a sublist.
Returns	The currentRecord.CurrentRecord object that called the method.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist.	2016.2

Parameter	Type	Required / Optional	Description	Since
			This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.cancelLine({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.commitLine(options)

Method Description	Commits the currently selected line on a sublist.
Returns	The <code>currentRecord.CurrentRecord</code> object that called the method.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.commitLine({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.findMatrixSublistLineWithValue(options)

Method Description	Returns the line number of the first instance where a specified value is found in a specified column of the matrix.
Returns	number
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
options.value	number	required	The value to search for.	2016.2
options.column	number	required	The column number of the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var lineNumber = objRecord.findMatrixSublistLineWithValue({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.findSublistLineWithValue(options)

Method Description	Returns the line number for the first occurrence of a field value in a sublist.
Returns	A line number as a number, or -1 if not found.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
options.value	number Date string array	optional	The value to search for.	2016.2

Parameter	Type	Required / Optional	Description	Since
	boolean true false			

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or not defined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var lineNumber = objRecord.findSublistLineWithValue({
  sublistId: 'item',
  fieldId: 'item',
  value: 233
});
...
//Add additional code
```

CurrentRecord.getCurrentMatrixSublistValue(options)

Method Description	Gets the value for the currently selected line in the matrix.
Returns	number Date string array boolean true false
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
options.column	number	required	The column number for the matrix field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var matrixValue = objRecord.getCurrentMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
//Add additional code
```

CurrentRecord.getCurrentSublistIndex(options)

Method Description	Returns the line number of the currently selected line.
Returns	number
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var currIndex = objRecord.getCurrentSublistIndex({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.getCurrentSublistSubrecord(options)

Method Description	Gets the subrecord for the associated sublist field on the current line. The subrecord object is retrieved in view mode.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var objSubrecord = objRecord.getCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item'
});
```

```
...
//Add additional code
```

CurrentRecord.getCurrentSublistText(options)

Method Description	Returns a text representation of the field value in the currently selected line.
Returns	string <div>  Note: For multiselect fields, returns an array. </div>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var fieldName = objRecord.getCurrentSublistText({
    sublistId: 'item',
    fieldId: 'item'
});
```

```
...
//Add additional code
```

CurrentRecord.getCurrentSublistValue(options)

Method Description	Returns the value of a sublist field on the currently selected sublist line.
Returns	number Date string array boolean true false
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var sublistValue = objRecord.getCurrentSublistValue({
    sublistId: 'item',
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.getField(options)

Method Description	Returns a field object from a record.
Returns	currentRecord.Field
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var objField = objRecord.getField({
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.getLineCount(options)

Method Description	Returns the number of lines in a sublist.
Returns	number
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None

Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var numLines = objRecord.getLineCount({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.getMatrixHeaderCount(options)

Method Description	Returns the number of columns for the specified matrix.
Returns	number
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var numLines = objRecord.getMatrixHeaderCount({
    sublistId: 'item',
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.getMatrixHeaderField(options)

Method Description	Gets the field for the specified header in the matrix.
Returns	currentRecord.Field
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2

Parameter	Type	Required / Optional	Description	Since
<code>options.column</code>	number	required	The column number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var objField = objRecord.getMatrixHeaderField({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
//Add additional code
```

CurrentRecord.getMatrixHeaderValue(options)

Method Description	Gets the value for the associated header in the matrix.
Returns	number Date string array boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2

Parameter	Type	Required / Optional	Description	Since
options.column	number	required	The column number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var value = objRecord.getMatrixHeaderValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
//Add additional code
```

CurrentRecord.getMatrixSublistField(options)

Method Description	Gets the field for the specified sublist in the matrix.
Returns	currentRecord.Field
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
options.column	number	required	The column number for the field.	2016.2
options.line	number	required	The line number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var objField = objRecord.getMatrixSublistField({
    sublistId: 'item',
    fieldId: 'item',
    column: 12,
    line: 3
});
...
//Add additional code
```

CurrentRecord.getMatrixSublistValue(options)

Method Description	Gets the value for the associated field in the matrix.
Returns	number Date string array boolean true false
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
options.column	number	required	The column number for the field.	2016.2
options.line	number	required	The line number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var value = objRecord.getMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12,
    line: 3
});
...
//Add additional code
```

CurrentRecord.getSublist(options)

Method Description	Returns the specified sublist.
Returns	currentRecord.Sublist
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
```

```

...
var objSublist = objRecord.getSublist({
    sublistId: 'item'
});
...
//Add additional code

```

CurrentRecord.getSublistField(options)

Method Description	Returns a field object from a sublist.
Returns	currentRecord.Field
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
options.line	number	required	The line number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
var objField = objRecord.getSublistField({
    sublistId: 'item',

```

```

    fieldId: 'item',
    line: 3
  });
  ...
  //Add additional code

```

CurrentRecord.getSublistText(options)

Method Description	Returns the value of a sublist field in a text representation.
Returns	string <div>  Note: For multiselect fields, returns an array. </div>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
options.line	number	required	The line number for the field.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...

```

```

var sublistFieldName = objRecord.getSublistText({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
//Add additional code

```

CurrentRecord.getSublistValue(options)

Method Description	Returns the value of a sublist field.
Returns	number Date string array boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
<code>options.line</code>	number	required	The line number for the field.	2016.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.
<code>SSS_INVALID_SUBLIST_OPERATION</code>	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
var sublistFieldValue = objRecord.getSublistValue({

```

```

        sublistId: 'item',
        fieldId: 'item',
        line: 3
    });
    ...
    //Add additional code

```

CurrentRecord.getSubrecord(options)

Method Description	Gets the subrecord associated with the field. The subrecord object is available in view mode.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
FIELD_1_IS_NOT_A_SUBRECORD_FIELD	The specified field is not a subrecord field.
FIELD_1_IS_DISABLED_YOU_CANNOT_APPLY_SUBRECORD_OPERATION_ON_THIS_FIELD	The specified field is disabled.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
var sublistFieldValue = objRecord.getSubrecord({
    fieldId: 'subrecord'

```



```
});
...
//Add additional code
```

CurrentRecord.getText(options)

Method Description	Returns the text representation of a field value.
Returns	string <div>  Note: For multiselect fields, returns an array. </div>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var fieldidname = objRecord.getText({
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.getValue(options)

Method Description	Returns the value of a field.
---------------------------	-------------------------------

Returns	number Date string array boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var value = objRecord.getValue({
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.hasCurrentSublistSubrecord(options)

Method Description	Returns a value indicating whether the associated sublist field has a subrecord on the current line. This method can only be used on dynamic records.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of a subrecord. See the help topic How do I find a field's internal ID?	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var hasSubrecord = objRecord.hasCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item'
});
...
//Add additional code
```

CurrentRecord.hasSublistSubrecord(options)

Method Description	Returns a value indicating whether the associated sublist field contains a subrecord.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a subrecord. See the help topic How do I find a field's internal ID?	2016.2
options.line	number	required	The line number for the field.	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var hasSubrecord = objRecord.hasSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
//Add additional code
```

CurrentRecord.hasSubrecord(options)

Method Description	Returns a value indicating whether the field contains a subrecord.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of the field that may contain a subrecord. See the help topic How do I find a field's internal ID?	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
```

```

var hasSubrecord = objRecord.hasSubrecord({
    fieldId: 'item'
});
...
//Add additional code

```

CurrentRecord.insertLine(options)

Method Description	Inserts a sublist line.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.line	number	required	The line number to insert.	2016.2
options.ignoreRecalc	boolean true false	optional	If set to true, scripting recalculation is ignored. The default value is false.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
objRecord.insertLine({
    sublistId: 'item',

```

```

    line: 3,
    ignoreRecalc: true
  });
  ...
  //Add additional code

```

CurrentRecord.removeCurrentSublistSubrecord(options)

Method Description	Removes the subrecord for the associated sublist field.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
objRecord.removeCurrentSublistSubrecord({
  sublistId: 'item',
  fieldId: 'item'
});
...
//Add additional code

```

CurrentRecord.removeLine(options)

Method Description	Removes a sublist line.
Returns	currentRecord.CurrentRecord

Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.line	number	required	The line number of the sublist to remove.	2016.2
options.ignoreRecalc	boolean true false	optional	If set to true, scripting recalculation is ignored. The default value is false.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.removeLine({
  sublistId: 'item',
  line: 3,
  ignoreRecalc: true
});
...
//Add additional code
```

CurrentRecord.removeSubrecord(options)

Method Description	Removes the subrecord for the associated field.
Returns	<code>currentRecord.CurrentRecord</code>

Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.removeSubrecord({
    fieldid: 'item'
});
...
//Add additional code
```

CurrentRecord.selectLine(options)

Method Description	Selects an existing line in a sublist.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist.	2016.2

Parameter	Type	Required / Optional	Description	Since
			This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	
options.line	number	required	The line number to select in the sublist.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var lineNum = objRecord.selectLine({
    sublistId: 'item',
    line: 3
});
...
//Add additional code
```

CurrentRecord.selectNewLine(options)

Method Description	Selects a new line at the end of a sublist.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.selectNewLine({
    sublistId: 'item'
});
...
//Add additional code
```

CurrentRecord.setCurrentMatrixSublistValue(options)

Method Description	Sets the value for the line currently selected in the matrix. This method is not available for standard records.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
options.column	number	required	The column number for the field.	2016.2

Parameter	Type	Required / Optional	Description	Since
options.value	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2016.2
options.ignoreFieldChange	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2
options.fireSlavingSync	boolean true false	optional	Indicates whether to perform field sourcing synchronously. If set to <code>true</code> , sources dependent field information for empty fields synchronously. Defaults to <code>false</code> – dependent field values are not sourced synchronously.	2016.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The options.value type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setCurrentMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 3,
    value: false,
    ignoreFieldChange: true,
    fireSlavingSync: true
});
...
//Add additional code
```

CurrentRecord.setCurrentSublistText(options)

Method Description	Sets the value for the field in the currently selected line by a text representation.
---------------------------	---

Returns	<code>currentRecord.CurrentRecord</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
<code>options.text</code>	string	required	The text to set the value to.	2016.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2
<code>options.fireSlavingSync</code>	boolean true false	optional	Indicates whether to perform field sourcing synchronously. If set to <code>true</code> , sources dependent field information for empty fields synchronously. Defaults to <code>false</code> – dependent field values are not sourced synchronously.	2016.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.
<code>A_SCRIPT_IS_ATTEMPTING_TO_EDIT_THE_1_SUBLIST_THIS_SUBLIST_IS_CURRENTLY_IN_READONLY_MODE_AND_CANNOT_BE_EDITED_CALL</code>	A read-only field is being edited.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
```

```

...
objRecord.setCurrentSublistText({
    sublistId: 'item',
    fieldId: 'item',
    text: 'value',
    ignoreFieldChange: true,
    fireSlavingSync: true
});
...
//Add additional code

```

CurrentRecord.setCurrentSublistValue(options)

Method Description	Sets the value for the field in the currently selected line.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See the help topic How do I find a field's internal ID?	2016.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2016.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2

Parameter	Type	Required / Optional	Description	Since
<code>options.forceSyncSourcing</code>	boolean true false	optional	Indicates whether to perform field sourcing synchronously. By default, this value is false.	2016.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The <code>options.value</code> type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
A_SCRIPT_IS_ATTEMPTING_TO_EDIT_THE_1_SUBLIST_THIS_SUBLIST_IS_CURRENTLY_IN_READ_ONLY_MODE_AND_CANNOT_BE_EDITED_CALL	ONLY MODES AND IF a read-only sublist field.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setCurrentSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    value: true,
    ignoreFieldChange: true
});
...
//Add additional code
```

CurrentRecord.setMatrixHeaderValue(options)

Method Description	Sets the value for the associated header in the matrix.
Returns	<code>currentRecord.CurrentRecord</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
<code>options.column</code>	number	required	The column number for the field.	2016.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2016.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2
<code>options.fireSlavingSync</code>	boolean true false	optional	Indicates whether to perform field sourcing synchronously. If set to <code>true</code> , sources dependent field information for empty fields synchronously. Defaults to <code>false</code> – dependent field values are not sourced synchronously.	2016.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The <code>options.value</code> type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setMatrixHeaderValue({
    sublistId: 'item',
```

```

    fieldId: 'item',
    column: 3,
    value: false,
    ignoreFieldChange: true,
    fireSlavingSync: true
  });
  ...
  //Add additional code

```

CurrentRecord.setMatrixSublistValue(options)

Method Description	Sets the value for the associated field in the matrix.
Returns	<code>currentRecord.CurrentRecord</code>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See the help topic How do I find a field's internal ID?	2016.2
<code>options.column</code>	number	required	The column number for the field.	2016.2
<code>options.line</code>	number	required	The line number for the field.	2016.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2016.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The <code>options.value</code> type does not match the field type.

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12,
    line: 3,
    value: true
});
...
//Add additional code
```

CurrentRecord.setText(options)

Method Description	Sets the value of the field by a text representation.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See the help topic How do I find a field's internal ID?	2016.2
options.text	string	required	The text to change the field value to.	2016.2
options.ignoreFieldChange	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2
options.fireSyncSlaving	boolean true false	optional	Indicates whether to perform field sourcing synchronously.	2018.2

Parameter	Type	Required / Optional	Description	Since
			If set to <code>true</code> , sources dependent field information for empty fields synchronously. Defaults to <code>false</code> – dependent field values are not sourced synchronously.	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setText({
    fieldId: 'item',
    text: 'value',
    ignoreFieldChange: true,
    fireSlavingSync: true
});
...
//Add additional code
```

CurrentRecord.setValue(options)

Method Description	Sets the value of a field.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field.	2016.2

Parameter	Type	Required / Optional	Description	Since
			See the help topic How do I find a field's internal ID?	
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio, Select and Multi-Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2016.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2016.2
<code>options.fireSyncSlaving</code>	boolean true false	optional	Indicates whether to perform field sourcing synchronously. If set to <code>true</code> , sources dependent field information for empty fields synchronously. Defaults to <code>false</code> – dependent field values are not sourced synchronously.	2018.2

Errors

Error Code	Thrown If
<code>INVALID_FLD_VALUE</code>	The <code>options.value</code> type does not match the field type.
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
objRecord.setValue({
    fieldId: 'item',
    value: true,
    ignoreFieldChange: true,
    fireSlavingSync: true
});
...
//Add additional code
```

CurrentRecord.id

Property Description	The internal ID of a specific record.
----------------------	---------------------------------------

Type	number (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var recordid = record.id;
...
//Add additional code
```

CurrentRecord.isDynamic

Property Description	Indicates whether the record is in dynamic mode. For more information, see SuiteScript 2.0 – Standard and Dynamic Modes . This value is set when the record is created or accessed.
Type	boolean <code>true</code> <code>false</code> (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (record.isDynamic) {
    ...
}
...
//Add additional code
```

CurrentRecord.type

Property Description	The current record's type. Note the following:
----------------------	---

	<ul style="list-style-type: none"> On an instance of a standard record type, this property is represented by a value from the record.Type enum. On an instance of a custom record type, this value is populated by the custom record type's string ID. For help finding this ID, see the help topic Custom Record.
Type	string (read-only)
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var recordtype = currentRecord.type;
...
//Add additional code
```

currentRecord.Field

Object Description	<p>Encapsulates a body or sublist field on the current record. Use the following methods to access the Field object:</p> <ul style="list-style-type: none"> CurrentRecord.getField(options) CurrentRecord.getSublistField(options) <p>For a complete list of this object's methods and properties, see N/currentRecord Module.</p>
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var currentRecordField = currentRecord.getField({
    fieldId: 'entity'
});
...
//Add additional code
}
```

Field.getSelectOptions(options)

Method Description	<p>Returns an array of available options on a standard or custom select, multiselect, or radio field as key-value pairs.</p> <div>  Important: You can use this method only in dynamic mode. For additional information on dynamic mode, see CurrentRecord.isDynamic. </div>
Returns	<p>array</p> <p>Only the first 1,000 available options are returned in an array. If there are more than 1,000 available options, an empty array [] is returned. This function returns an array in the following format:</p> <pre>[{value: 5, text: 'abc'}, {value: 6, text: '123'}]</pre> <p>This function returns <code>Type Error</code> if the field is not a supported field for this method.</p>
Governance	None
Supported Script Types	<p>Client scripts</p> <p>For more information, see SuiteScript 2.0 Client Script Type.</p>
Module	N/currentRecord Module
Since	2016.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.filter</code>	string	Required	<p>The search string to filter the select options that are returned.</p> <div>  Note: Filter values are case insensitive. </div>	2016.2
<code>options.operator</code>	string	Required	<p>The following operators are supported:</p> <ul style="list-style-type: none"> contains (default) is startswith 	2016.2

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/currentRecord Module Script Sample .
--

```
//Add additional code
...
var options = objField.getSelectOptions({
  filter : 'C',
  operator : 'startswith'
});
```

```
...
//Add additional code
```

Field.insertSelectOption(options)

Method Description	Inserts an option into certain types of select and multiselect fields. This method is usable only in select and multiselect fields that were added by a front-end Suitelet or beforeLoad user event script. The IDs for these fields always have a prefix of custpage .
Returns	Void
Governance	None
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.value	string	Required	A string, not visible in the UI, that identifies the option.	2016.2
options.text	string	Required	The label that represents the option in the UI.	2016.2
options.isSelected	boolean	Optional	Determines whether the option is selected by default. If not specified, this value defaults to false.	2016.2

Errors

Error Code	Thrown If
SSS_INVALID_UI_OBJECT_TYPE	A script attempts to use this method on the wrong type of field. This method can be used only on select and multiselect fields whose IDs begin with the prefix custpage .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example.

```
//Add additional code
...

// Instantiate the field. Note that this method is supported only
// on fields whose fieldIds have a prefix of custpage.

var field = call.getField({
```

```

        fieldId: 'custpage_select1field'
    });

    // Insert a new option.

    field.insertSelectOption({
        value: 'Option1',
        text: 'alpha'
    });
    ...
    //Add additional code

```

Field.removeSelectOption(options)

Method Description	Removes a select option from certain types of select and multiselect fields. This method is usable only in select fields that were added by a front-end Suitelet or beforeLoad user event script. The IDs for these fields always have a prefix of custpage .
Returns	Void
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.value	string	Required	<p>A string, not shown in the UI, that identifies the option. To remove all options from the list, set this field to null, as follows:</p> <pre> ... field.removeSelectOption({ value: null, }); ... </pre>	2016.2

Errors

Error Code	Thrown If
SSS_INVALID_UI_OBJECT_TYPE	A script attempts to use this method on the wrong type of field. This method can be used only on select and multiselect fields whose IDs begin with the prefix custpage .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example.

```
//Add additional code
...

// Instantiate the field. Note that this method is supported only
// on fields whose fieldIds have a prefix of custpage.

var field = call.getField({
    fieldId: 'custpage_select1field'
});

// Remove the appropriate option.

field.removeSelectOption({
    value: 'Option2',
});

...
//Add additional code
```

Field.id

Property Description	Returns the internal ID of a standard or custom body or sublist field.
Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var id = objField.id;
...
//Add additional code
```

Field.label

Property Description	Returns the UI label for a standard or custom field body or sublist field.
-----------------------------	--

Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var label = objField.label;
...
//Add additional code
```

Field.isMandatory

Property Description	Returns <code>true</code> if the standard or custom field is mandatory on the record form, or <code>false</code> otherwise.
Type	boolean <code>true</code> <code>false</code>
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objField.isMandatory) {
    ...
}
...
//Add additional code
```

Field.isDisabled

Property Description	This property reflects the display type of a field. A value of <code>true</code> means the field is disabled. A value of <code>false</code> means the field is enabled. Note also:
----------------------	--

	<ul style="list-style-type: none"> ■ If you are working with a body field, you can use this property to change the field's display type. ■ If you are working with a sublist field, you can set this property to <code>true</code> or <code>false</code>, but be aware that this action affects the entire sublist column, even though a sublist field is associated with one line. ■ For both body and sublist fields, you can use <code>Field.isDisabled</code> to determine whether the field is disabled or enabled.
Type	boolean <code>true</code> <code>false</code>
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objField.isDisabled) {
    ...
}
...
//Add additional code
```

Field.isPopup

Property Description	Returns <code>true</code> if the field is a popup list field, or <code>false</code> otherwise.
Type	boolean <code>true</code> <code>false</code> (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objField.isPopup) {
    ...
}
...
```

```
//Add additional code
```

Field.isDisplay

Property Description	Returns <code>true</code> if the field is set to display on the record form, or <code>false</code> otherwise. Fields can be a part of a record even if they are not displayed on the record form. This property is read-only for sublist fields.
Type	boolean <code>true</code> <code>false</code>
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objField.isDisplay) {
    ...
}
...
//Add additional code
```

Field.isVisible

Property Description	Returns <code>true</code> if the field is visible on the record form, or <code>false</code> otherwise.
Type	boolean <code>true</code> <code>false</code> (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objField.isVisible) {
    ...
}
```

```

}
...
//Add additional code

```

Field.isReadOnly

Property Description	Returns <code>true</code> if the field on the record form cannot be edited, or <code>false</code> otherwise. For textarea fields, this property can be read or written to. For all other fields, this property is read-only.
Type	boolean <code>true</code> <code>false</code>
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
if (objField.isReadOnly) {
    ...
}
...
//Add additional code

```

Field.sublistId

Property Description	Returns the sublist ID for the specified sublist field.
Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...

```

```
var myId = field.sublistId;
...
//Add additional code
```

Field.type

Property Description	Returns the type of a body or sublist field. For example, the value can return <code>text</code> , <code>date</code> , <code>currency</code> , <code>select</code> , <code>checkbox</code> , and other similar values.
Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var type = objField.type;
...
//Add additional code
```

currentRecord.Sublist

Object Description	Encapsulates a sublist on the current record. For a complete list of this object's methods and properties, see N/currentRecord Module .
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/currentRecord Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var objSublist = currentRecord.getSublist({
```

```

        sublistId: 'item'
    });
    ...
    //Add additional code

```

Sublist.getColumn(options)

Method Description	Returns a column in the sublist.
Returns	currentRecord.Column
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of the column field in the sublist. See the help topic How do I find a field's internal ID?	2016.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```

//Add additional code
...
var objColumn = objSublist.getColumn({
    fieldId: 'item'
});
...
//Add additional code

```

Sublist.id

Property Description	Returns the internal ID of the sublist.
Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts

	For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var sublistid = objSublist.id;
...
//Add additional code
```

Sublist.isChanged

Property Description	Indicates whether the sublist has changed on the current record form.
Type	boolean <code>true</code> <code>false</code> (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objSublist.isChanged) {
    ...
}
...
//Add additional code
```

Sublist.isDisplay

Property Description	Indicates whether the sublist is displayed on the current record form.
Type	boolean <code>true</code> <code>false</code> (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .

Since	2016.2
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
if (objSublist.isDisplay) {
    ...
}
...
//Add additional code
```

Sublist.type

Property Description	Returns the sublist type.
Type	string (read-only)
Module	N/currentRecord Module
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var sublisttype = objSublist.type;
...
//Add additional code
```

currentRecord.get()

Method Description	Retrieves a currentRecord object that represents the record active on the current page.
Returns	currentRecord.CurrentRecord
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module

Since	2016.2
-------	--------

Errors

Error Code	Thrown If
CANNOT_CREATE_RECORD_INSTANCE	The current record page is not scriptable or an error occurred when creating the record object.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var record = currentRecord.get();
...
//Add additional code
```

currentRecord.get.promise()

Method Description	Retrieves a promise for a currentRecord object that represents the record active on the current page.
Returns	Promise
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/currentRecord Module
Since	2016.2

Errors

Error Code	Thrown If
CANNOT_CREATE_RECORD_INSTANCE	The current record page is not scriptable or an error occurred when creating the record instance.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/currentRecord Module Script Sample](#).

```
//Add additional code
...
var record = currentRecord.get.promise();
```

```
...
//Add additional code
```

N/email Module

Load the N/email module when you want to send email messages from within NetSuite. You can use the N/email module to send regular, bulk, and campaign email.

- [N/email Module Members](#)
- [N/email Module Script Sample](#)

N/email Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	email.send(options)	void	Client and server-side scripts	Sends email to an individual or group of recipients and receives bounceback notifications.
	email.send.promise(options)	void	Client and server-side scripts	Sends email asynchronously to an individual or group of recipients and receives bounceback notifications.
	email.sendBulk(options)	void	Client and server-side scripts	Sends bulk email.
	email.sendBulk.promise(options)	void	Client and server-side scripts	Sends bulk email asynchronously.
	email.sendCampaignEvent(options)	number	Client and server-side scripts	Sends lead nurturing campaigns (drip marketing email).
	email.sendCampaignEvent.promise(options)	number	Client and server-side scripts	Sends lead nurturing campaigns (drip marketing email) asynchronously.

N/email Module Script Sample

The following example sends email with an attachment.

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

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

```

*/
require(['N/email', 'N/record', 'N/file'],
function(email, record, file) {
    function sendEmailWithAttachement() {
        var senderId = -5;
        var recipientEmail = 'notify@myCompany.com';
        var timeStamp = new Date().getUTCMilliseconds();
        var 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
        });
        var recipientId = recipient.save();
        var 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 //an integer value
                }
            }
        });
    }
    sendEmailWithAttachement();
});

```

email.send(options)

Method Description	<p>Method used to send transactional email and receive bounceback notifications if the email is not successfully delivered.</p> <p>A maximum of 10 recipients (recipient + cc + bcc) is allowed.</p> <p>The total message size (including attachments) must be 15MB or less. The size of Individual attachments cannot exceed 10BM.</p>
Returns	void

Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	20 usage units
Module	N/email Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.author</code>	number	required	<ul style="list-style-type: none"> Internal ID of the email sender. To find the internal ID of the sender in the UI, go to Lists > Employees. 	2015.2
<code>options.recipients</code>	number[] string[]	required	<ul style="list-style-type: none"> The internal ID or email address of the recipient. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. <div>  Note: Only the first recipient displays on the Communication tab (under the Recipient column). To view all recipients, click View to open the Message record. </div>	2015.2
<code>options.replyTo</code>	string	optional	<ul style="list-style-type: none"> The email address that appears in the reply-to header when an email is sent out. You can use either a single external email address or a generic email address created by the Email Capture Plug-in. 	2015.2
<code>options.cc</code>	number[] string[]	optional	<ul style="list-style-type: none"> The internal ID or email address of the secondary recipient to copy. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. 	2015.2
<code>options.bcc</code>	number[] string[]	optional	<ul style="list-style-type: none"> The internal ID or email address of the recipient to blind copy. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. 	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.subject</code>	string	required	<ul style="list-style-type: none"> Subject of the outgoing message. 	2015.2
<code>options.body</code>	string	required	<ul style="list-style-type: none"> Contents of the email SuiteScript formats the body of the email in either plain text or HTML. If HTML tags are present, the message is formatted as HTML. Otherwise, the message is formatted in plain text. To display XML as plain text, use an <code>HTML<pre></code> tag around XML content. 	2015.2
<code>options.attachments</code>	file.File	optional	<ul style="list-style-type: none"> The email file attachments. You can send multiple attachments of any media type An individual attachment must not exceed 10MB and the total message size must be 15MB or less. <div>  Note: Supported for server-side scripts only. </div>	2015.2
<code>options.relatedRecords</code>	Object	optional	<ul style="list-style-type: none"> Object that contains key/value pairs to associate the Message record with related records (including custom records). See the relatedRecords table for more information 	2015.2
<code>options.isInternalOnly</code>	boolean true false	optional	<ul style="list-style-type: none"> If true, the Message record is not visible to an external Entity (for example, a customer or contact). The default value is <code>false</code>. 	2015.2

 **Note:** The `relatedRecords` parameter is a JavaScript object and the table below lists its properties.

relatedRecords

You can associate the sent email with an array of internal records using key/value pairs.

Parameter	Type	Required / Optional	Description	Since
<code>transactionId</code>	number	optional	The Transaction record(s) associated with the Message record. Use for transaction and opportunity record types.	2015.2
<code>activityId</code>	number	optional	The Activity record(s) attached to the Email Message record Use for Case and Campaign record types.	2015.2
<code>entityId</code>	number	optional	The Entity record(s) attached to the Email Message record Use for all Entity record types (for example, customer, contact)	2015.2

Parameter	Type	Required / Optional	Description	Since
customRecord	Object	optional	The custom record(s) attached to the Email Message record For custom records you must specify both the record ID and the record type ID.	2015.2
customRecord.id	number	optional	The instance ID for the custom record attached to the Email Message record Note: If you use this parameter, <code>customRecord.recordType</code> is required.	2015.2
customRecord.recordType	string	optional	The integer ID for the custom record type attached to the Message record. This ID is shown as part of the record's URL. For example: <code>/custrecordentry.nl?rectype=2&id=56</code> Note: If you use this parameter, <code>customRecord.id</code> is required.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/email Module Script Sample](#).

```
//Add additional code
...
var senderId = -5;
var recipientEmail = 'notify@myCompany.com';
var timeStamp = new Date().getUTCMilliseconds();
var 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
});
var recipientId = recipient.save();
var 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 //an integer value
  }
}
});
...
//Add additional code

```

email.send.promise(options)

Method Description	<p>Method used to send transactional email asynchronously and receive bounceback notifications if the email is not successfully delivered.</p> <div>  Note: For information about the parameters and errors thrown for this method, see email.send(options). For additional information about promises, see Promise Object. </div>
Returns	void
Synchronous Version	email.send(options)
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	20 usage units
Module	N/email Module
Since	2015.2

email.sendBulk(options)

Method Description	<p>This method is used to send bulk email when a bounceback notification is not required.</p> <div>  Note: This API normally uses a bulk email server to send messages. If you need to increase the successful delivery rate of an email, use email.send(options) so that a transactional email server is used. </div>
Returns	void
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/email Module

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.author	number	required	<ul style="list-style-type: none"> Internal ID of the email sender. To find the internal ID of the sender in the UI, go to Lists > Employees. 	2015.2
options.recipients	number[] string[]	required	<ul style="list-style-type: none"> The internal ID or email address of the recipient. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. <div>  Note: Only the first recipient displays on the Communication tab (under the Recipient column). To view all recipients, click View to open the Message record. </div>	2015.2
options.replyTo	string	optional	<ul style="list-style-type: none"> The email address that appears in the reply-to header when an email is sent out. You can use either a single external email address or a generic email address created by the Email Capture Plug-in. 	2015.2
options.cc	number[] string[]	optional	<ul style="list-style-type: none"> The internal ID or email address of the secondary recipient to copy. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. 	2015.2
options.bcc	number[] string[]	optional	<ul style="list-style-type: none"> The internal ID or email address of the recipient to blind copy. For multiple recipients, use an array of internal IDs or email addresses. You can use an array that contains a combination of internal IDs and email addresses. A maximum of 10 recipients (recipient + cc + bcc) is allowed. 	2015.2
options.subject	string	required	<ul style="list-style-type: none"> Subject of the outgoing message. 	2015.2
options.body	string	required	<ul style="list-style-type: none"> Contents of the email SuiteScript formats the body of the email in either plain text or HTML. If HTML tags are present, the message is formatted as HTML. 	2015.2

Parameter	Type	Required / Optional	Description	Since
			Otherwise, the message is formatted in plain text. To display XML as plain text, use an HTML<pre> tag around XML content.	
options.attachments	file.File	optional	<ul style="list-style-type: none"> The email file attachments. You can send multiple attachments of any media type An individual attachment must not exceed 10MB and the total message size must be 15MB or less. <div>  Note: Supported for server-side scripts only. </div>	2015.2
options.relatedRecords	Object	optional	<ul style="list-style-type: none"> Object that contains key/value pairs to associate the Message record with related records (including custom records). See the relatedRecords table for more information 	2015.2
options.isInternalOnly	boolean true false	optional	<ul style="list-style-type: none"> If true, the Message record is not visible to an external Entity (for example, a customer or contact). The default value is <code>false</code>. 	2015.2

 **Note:** The `relatedRecords` parameter is a JavaScript object and the table below lists its properties.

relatedRecords

You can associate the sent email with an array of internal records using key/value pairs.

Parameter	Type	Required / Optional	Description	Since
transactionId	number	optional	<ul style="list-style-type: none"> The Transaction record(s) attached to the Message record Use for transaction and opportunity record types. 	2015.2
activityId	number	optional	<ul style="list-style-type: none"> The Activity record(s) attached to the Email Message record Use for Case and Campaign record types. 	2015.2
entityId	number	optional	<ul style="list-style-type: none"> The Entity record(s) attached to the Email Message record Use for all Entity record types (for example, customer, contact) 	2015.2
customRecord	Object	optional	<ul style="list-style-type: none"> The custom record(s) attached to the Email Message record For custom records you must specify both the record ID and the record type ID. 	2015.2

Parameter	Type	Required / Optional	Description	Since
customRecord.id	number	optional	<ul style="list-style-type: none"> The instance ID for the custom record attached to the Email Message record 	2015.2
customRecord.recordType	string	optional	<ul style="list-style-type: none"> The custom record type attached to the Message record 	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/email Module Script Sample](#).

```
//Add additional code
...
var recipientEmails = [
    'msample@netsuite.com',
    'jdoe@netsuite.com',
    'awolfe@netsuite.com',
    'htest@netsuite.com'
];
email.sendBulk({
    author: -5,
    recipients: recipientEmails,
    subject: 'Order Status',
    body: 'Your order has been completed.',
    replyTo: 'accounts@netsuite.com'
});
...
//Add additional code
```

email.sendBulk.promise(options)

Method Description	<p>This method is used to send bulk email asynchronously when a bounceback notification is not required.</p> <div> Note: For information about the parameters and errors thrown for this method, see email.sendBulk(options). For additional information about promises, see Promise Object. </div>
Returns	void
Synchronous Version	email.sendBulk(options)
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/email Module
Since	2015.2

email.sendCampaignEvent(options)

Method Description	<p>Method used to send a single “on-demand” campaign email to a specified recipient and return a campaign response ID to track the email.</p> <p>Email (campaignemail) sublists are not supported. The campaign must use a Lead Nurturing (campaigndrip) sublist.</p> <div>  Note: This API normally uses a bulk email server to send messages. If you need to increase the successful delivery rate of an email, use <code>email.send(options)</code> so that a transactional email server is used. </div>
Returns	A campaign response ID (tracking code) as number If the email fails to send, the value returned is -1.
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/email Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.campaignEventId</code>	number	required	<ul style="list-style-type: none"> The internal ID of the campaign event. <div>  Note: The campaign must use a Lead Nurturing (campaigndrip) sublist. </div>	2015.2
<code>options.recipientId</code>	number	required	<ul style="list-style-type: none"> The internal ID of the recipient. <div>  Note: The recipient's record must contain an email address. </div>	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/email Module Script Sample](#).

```
//Add additional code
...
email.sendCampaignEvent({
    campaignEventId: -8,
    recipientId: 142,
});
...
//Add additional code
```

email.sendCampaignEvent.promise(options)

Method Description	Method used to send a single “on-demand” campaign email asynchronously to a specified recipient and return a campaign response ID to track the email. <div>  Note: For information about the parameters and errors thrown for this method, see email.sendCampaignEvent(options). For additional information about promises, see Promise Object. </div>
Returns	A campaign response ID (tracking code) as a number. If the email fails to send, the value returned is -1.
Synchronous Version	email.sendCampaignEvent(options)
Supported Script Types	Client and server-side scripts For additional information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/email Module
Since	2015.2

N/encode Module

This module exposes string encoding and decoding functionality. Load the N/encode module when you want to convert a string to another type of encoding.

- [N/encode Module Members](#)
- [N/encode Module Script Sample](#)

N/encode Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	encode.convert(options)	string	Server-side scripts	Converts a string to another type of encoding and returns the re-encoded string.
Enum	encode.Encoding	enum	Server-side scripts	Holds the string values for supported encoding specifications.

N/encode Module Script Sample

The following example converts a string to a different encoding.

 **Note:** This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
```

```

* @NapiVersion 2.x
*/
require(['N/encode'],
function (encode) {
  function convertStringToDifferentEncoding() {
    var stringInput = "TAfA(c)st StriAfA+~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();
});

```

encode.convert(options)

Method Description	Converts a string to another type of encoding.
Returns	The re-encoded string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/encode Module
Since	2015.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.string	string	required	The string to encode.	2015.1
options.inputEncoding	string	required	The encoding used on the input string. The default value is <code>UTF_8</code> . Use the <code>encode.Encoding</code> to set the value.	2015.1
options.outputEncoding	string	required	The encoding to apply to the output string. The default value is <code>UTF_8</code> . Use the <code>encode.Encoding</code> to set the value.	2015.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/encode Module Script Sample](#).

```
//Add additional code
```

```

...
var hexEncodedString = encode.convert({
    string: stringInput,
    inputEncoding: encode.Encoding.UTF_8,
    outputEncoding: encode.Encoding.HEX
});
...
//Add additional code

```

encode.Encoding

Enum Description	<p>Holds the string values for the supported character set encoding. This enum is used to set the value of <code>inputEncoding</code> and <code>outputEncoding</code> parameters that are members of the N/crypto Module or N/encode Module.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/encode Module
Since	2015.1

Values

- UTF_8
- BASE_16
- BASE_32
- BASE_64
- BASE_64_URL_SAFE
- HEX

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/encode Module Script Sample](#).

```

//Add additional code
...
var reencoded = encode.convert({
    string: LOREM_IPS,
    inputEncoding: encode.Encoding.BASE_64,
    outputEncoding: encode.Encoding.UTF_8
});
...
//Add additional code

```

N/error Module

Load the error module when you want to create your own custom SuiteScript errors. Use these custom errors in try-catch statements to abort script execution.

- [N/error Module Members](#)
- [SuiteScriptError Object Members](#)
- [UserEventError Object Members](#)
- [N/error Module Script Samples](#)

N/error Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	error.SuiteScriptError	Object	Server-side scripts that are not user event scripts	Encapsulates a SuiteScript error thrown by any script type that is not a user event script.
	error.UserEventError	Object	User event scripts	Encapsulates a SuiteScript error thrown by a user event script.
Method	error.create(options)	error.SuiteScriptError or error.UserEventError	Server-side scripts	Creates a new error.SuiteScriptError or error.UserEventError object.

SuiteScriptError Object Members

The following members are called on [error.SuiteScriptError](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	SuiteScriptError.name	string (read-only)	Server-side scripts that are not user event scripts	User-defined error code.
	SuiteScriptError.message	string (read-only)	Server-side scripts that are not user event scripts	Text that displays on the SuiteScript Execution Log, in the Details column.
	SuiteScriptError.id	string (read-only)	Server-side scripts that are not user event scripts	Error ID that is automatically generated when a new error is created.
	SuiteScriptError.stack	Array of strings (read-only)	Server-side scripts that are not user event scripts	A list of method calls that the script is executing when the error is thrown.

UserEventError Object Members

The following members are called on [error.UserEventError](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	UserEventError.name	string (read-only)	User event scripts	User-defined error code.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	UserEventError.message	string (read-only)	User event scripts	Text that displays on the SuiteScript Execution Log, in the Details column.
	UserEventError.eventType	string (read-only)	User event scripts	User event type (beforeLoad, beforeSubmit, afterSubmit)
	UserEventError.id	string (read-only)	User event scripts	Error ID that is automatically generated when a new error is created.
	UserEventError.recordId	string (read-only)	User event scripts	Internal ID of the submitted record that triggered the script. This property only holds a value when the error is thrown by an afterSubmit user event script.
	UserEventError.stack	Array of strings (read-only)	User event scripts	A list of method calls that the script is executing when the error is thrown.

N/error Module Script Samples

The following example creates an error.

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 *@NApiVersion 2.x
 */
require(['N/error'],
function(error) {
    function createError() {
        var errorObj = error.create({
            name: 'MY_CODE',
            message: 'my error details',
            notifyOff: true
        });
    }
    createError();
});
```

The following example creates an error if the variable `somevariable` is false. In the `createError()` function's condition is met, the error is logged and then thrown — the script's execution results in an error being thrown.

```
/**
 *@NApiVersion 2.x
 */
require(['N/error'],
```

```

function(error) {
    function showError() {
        var somevariable = false;

        if (!somevariable) {
            var errObj = error.create({name : error.Type.WRONG_PARAMETER_TYPE, message : 'Wrong parameter type
selected.', notifyOff: false});
            log.error('Error: ' + errObj.name , errObj.message);
            throw errObj;
        }
    }
    showError();
});

```

error.SuiteScriptError

Object Description	<p>Encapsulates a SuiteScript error for any script type that is not a user event script. Use this object in a try-catch statement to abort script execution. Create a new custom error (<code>error.SuiteScriptError</code>) with the <code>error.create(options)</code> method. The <code>error.create(options)</code> method returns <code>error.SuiteScriptError</code> when it is called in any server-side script that is not a user event script.</p> <div>  Note: When <code>error.create(options)</code> is called in a user event script, it returns <code>error.UserEventError</code>. </div> <p>For a complete list of this object's methods and properties, see SuiteScriptError Object Members.</p>
Supported Script Types	All server-side scripts that are not user event scripts. For more information, see SuiteScript 2.0 Script Types .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```

//Add additional code
...
var errorObj = error.create({
    name: 'MY_CODE',
    message: 'my error details',
    notifyOff: false
});
...
//Add additional code

```

SuiteScriptError.id

Property Description	Error ID that is automatically generated when a new error is created.
-----------------------------	---

	This property is read-only.
Type	string
Supported Script Types	All server-side scripts that are not user event scripts. For more information, see SuiteScript 2.0 Script Types .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error ID: " + errorObj.id);
...
//Add additional code
```

SuiteScriptError.message

Property Description	Text that displays on the SuiteScript Execution Log, in the Details column. This property is read-only.
Type	string
Supported Script Types	All server-side scripts that are not user event scripts. For more information, see SuiteScript 2.0 Script Types .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error Message: " + errorObj.message);
...
//Add additional code
```

SuiteScriptError.name

Property Description	A user-defined name (error code). This property is read-only.
Type	string
Supported Script Types	All server-side scripts that are not user event scripts. For more information, see SuiteScript 2.0 Script Types .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error Code: " + errorObj.name);
...
//Add additional code
```

SuiteScriptError.stack

Property Description	A list of method calls that the script is executing when the error is thrown. The most recently executed method is listed at the top. This property is read-only.
Type	Array of strings
Supported Script Types	All server-side scripts that are not user event scripts. For more information, see SuiteScript 2.0 Script Types .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
```

```
});
log.debug("Error Stack: " + errorObj.stack);
...
//Add additional code
```

error.UserEventError

Object Description	<p>Encapsulates a SuiteScript error for user event scripts. Use this object in a try-catch statement to abort script execution. Create a new custom error (<code>error.UserEventError</code>) with the <code>error.create(options)</code> method. The <code>error.create(options)</code> method returns <code>error.UserEventError</code> when it is called in a user event script.</p> <div>  Note: When <code>error.create(options)</code> is called in a server-side script that is not a user event script, it returns <code>error.SuiteScriptError</code>. </div> <p>For a complete list of this object's methods and properties, see UserEventError Object Members.</p>
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
...
//Add additional code
```

UserEventError.eventType

Property Description	<p>The user event type. Holds one of the following values:</p> <ul style="list-style-type: none"> beforeLoad beforeSubmit afterSubmit <p>This property is read-only.</p>
Type	string
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .

Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
    name: 'MY_CODE',
    message: 'my error details',
    notifyOff: false
});
log.debug("User Event Type: " + errorObj.eventType);
...
//Add additional code
```

UserEventError.stack

Property Description	A list of method calls that the script is executing when the error is thrown. The most recently executed method is listed at the top. This property is read-only.
Type	Array of strings
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
    name: 'MY_CODE',
    message: 'my error details',
    notifyOff: false
});
log.debug("Error Stack: " + errorObj.stack);
...
//Add additional code
```

UserEventError.id

Property Description	Error ID that is automatically generated when a new error is created.
----------------------	---

	This property is read-only.
Type	string
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error ID: " + errorObj.id);
...
//Add additional code
```

UserEventError.message

Property Description	Text that displays on the SuiteScript Execution Log, in the Details column. This property is read-only.
Type	string
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error Message: " + errorObj.message);
...
//Add additional code
```

UserEventError.name

Property Description	A user-defined name (error code). This property is read-only.
Type	string
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error Code: " + errorObj.name);
...
//Add additional code
```

UserEventError.recordId

Property Description	The internal ID of the submitted record that triggered the script. This property only holds a value when the error is thrown by an afterSubmit user event script. This property is read-only.
Type	string
Supported Script Types	User event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Module	N/error Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
```



```
log.debug("Submitted Record ID: " + errorObj.recordId);
...
//Add additional code
```

error.create(options)

Method Description	Method used to create a new error.SuiteScriptError or error.UserEventError object. Use this custom error in a try-catch statement to abort script execution.
Returns	One of the following: <ul style="list-style-type: none"> An error.UserEventError object if the script throwing the error is a user event script. An error.SuiteScriptError object if the script throwing the error is any other server-side script.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/error Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object. The table below describes the name:value pairs that make up the object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	<ul style="list-style-type: none"> A user-defined name (error code). Sets the value for the property SuiteScriptError.name or UserEventError.name. 	2015.2
options.message	string	required	<ul style="list-style-type: none"> The error message displayed. This value displays on the Execution Log, in the Details column. The default value is null. Sets the value for the property SuiteScriptError.message or UserEventError.message. <pre>//test var errorObj = error.create({ name: 'MY_CODE', message: 'my error details', notifyOff: false }); log.debug("Error Code: " + errorObj.name);</pre>	2015.2
options.notifyOff	boolean true false	optional	<ul style="list-style-type: none"> Sets whether email notification is suppressed. The default value is false. If set to false, when this error is thrown, the system emails the users identified on the applicable script record's Unhandled Errors subtab. For additional information on the Unhandled Errors subtab, see Creating a Script Record. 	2015.2

Errors

Error Code	Message	Thrown If
MISS_MANDATORY_PARAMETER		A required argument is missing

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/error Module Script Samples](#).

```
//Add additional code
...
var errorObj = error.create({
  name: 'MY_CODE',
  message: 'my error details',
  notifyOff: false
});
log.debug("Error Code: " + errorObj.name);
...
//Add additional code
```

N/file Module

Load the file module when you want to work with files within NetSuite. You can use this module to upload files to the NetSuite file cabinet. You can also use this module to send files as attachments without uploading them to the file cabinet.

Methods that load content in memory, such as [File.getContents\(\)](#), have a 10MB size limit. This limit does not apply when content is streamed, such as when [File.save\(\)](#) is called.

- [N/file Module Members](#)
- [File Object Members](#)
- [N/file Module Script Sample](#)

N/file Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	file.File	object	Server-side scripts	Encapsulates a file within NetSuite.
Method	file.create(options)	file.File	Server-side scripts	Creates a new file.File .
	file.delete(options)	void	Server-side scripts	Deletes an existing file.File from the NetSuite file cabinet.
	file.load(options)	file.File	Server-side scripts	Loads an existing file.File from the NetSuite file cabinet.
Enum	file.Encoding	enum	Server-side scripts	Sets the value of the File.encoding property.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>file.Type</code>	enum	Server-side scripts	Sets the value of the <code>File.fileType</code> property.

File Object Members

The following members are called on `file.File`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>File.appendLine(options)</code>	<code>file.File</code> Object	Server-side scripts	Inserts a line to the end of a CSV or text file.
	<code>File.getContents()</code>	string	Server-side scripts	Returns the content of a file in string format.
	<code>File.lines.iterator()</code>	boolean <code>true</code> <code>false</code>	Server-side scripts	Calls a developer-defined function for each line. Returns false when line processing stops.
	<code>File.resetStream()</code>	void	Server-side scripts	Resets the file stream to its previous state.
	<code>File.save()</code>	number	Server-side scripts	Saves a new or updated file to the file cabinet.
Property	<code>File.description</code>	string	Server-side scripts	Description of a file.
	<code>File.encoding</code>	string	Server-side scripts	Character encoding on a file.
	<code>File.fileType</code>	enum	Server-side scripts	File type of a file.
	<code>File.folder</code>	number	Server-side scripts	Internal ID of the folder that houses a file within the NetSuite file cabinet.
	<code>File.id</code>	number (read-only)	Server-side scripts	Internal ID of a file in the NetSuite file cabinet.
	<code>File.isInactive</code>	boolean <code>true</code> <code>false</code>	Server-side scripts	Inactive status of a file. If set to true, the file is inactive.
	<code>File.isOnline</code>	boolean <code>true</code> <code>false</code>	Server-side scripts	“Available without Login” status of a file. If set to true, users can download the file outside of a current NetSuite login session.
	<code>File.isText</code>	boolean (read-only)	Server-side scripts	Indicates whether a file type is text-based.
	<code>File.name</code>	string	Server-side scripts	Name of a file.
	<code>File.path</code>	string (read-only)	Server-side scripts	Relative path to a file in the NetSuite file cabinet.
	<code>File.size</code>	number (read-only)	Server-side scripts	Size of a file in bytes.
	<code>File.url</code>	string (read-only)	Server-side scripts	URL of a file.

N/file Module Script Sample

Note: These sample scripts use the `require` function so that you can copy into the debugger and test. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Example 1

The following example creates and saves a file to the file cabinet.

Warning: In this example, the folder ID value is hard-coded. For the sample code to run in the SuiteScript Debugger, you must replace this hard-coded value with a valid folder ID from your account.

```
/**
 *@NApiVersion 2.x
 */
require(['N/file'],
function(file) {
    function createAndSaveFile() {
        var fileObj = file.create({
            name: 'test.txt',
            fileType: file.Type.PLAINTEXT,
            contents: 'Hello World\nHello World'
        });
        fileObj.folder = -15;
        var id = fileObj.save();
        fileObj = file.load({
            id: id
        });
    }
    createAndSaveFile();
});
```

Example 2

The following sample creates and saves a file to the file cabinet. It also sets the values of `File.isOnline` and the `File.folder` properties.

Warning: In this example, the folder ID value is hard-coded. For the sample code to run in the SuiteScript Debugger, you must replace this hard-coded value with a valid folder ID from your account.

```
/**
 *@NApiVersion 2.x
 */
require(['N/file'],
function(file){
    function createAndSaveFile(){
        var fileObj = file.create({
```

```

        name: 'test.txt',
        fileType: file.Type.PLAINTEXT,
        contents: 'Hello World\nHello World',
        folder : -15,
        isOnline : true
    });

    var id = fileObj.save();
    fileObj = file.load({
        id: id
    });
}

createAndSaveFile();
}
);

```

Example 3

```

require(['N/file', 'N/error', 'N/log'],
function (file, error, log)
{

    // In this sample we will compute the total for the
    // second column value in a csv file.
    //
    // date,amount
    // 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'
    });
    csvFile.appendLine({
        value: '10/21/14,200.0'
    });
    csvFile.appendLine({
        value: '10/21/15,210.2'
    });
    csvFile.appendLine({
        value: '10/21/16,250.3'
    });
    var csvFileId = csvFile.save();

    // This variable will store the total.
    var total = 0.0;

    // Load the file and
    // process all the lines

```

```

var invoiceFile = file.load({
    id: csvFileId
});
var iterator = invoiceFile.lines.iterator();

//Skip the first line (CSV header)
iterator.each(function () {return false;});
iterator.each(function (line)
{
    // This function updates the total by
    // adding the amount on each line to it
    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;
});

// By the time you are here, the total variable is
// set to 660.5
log.debug({
    title: 'total',
    details: total
});
}
)

```

file.File

Object Description	<p>Encapsulates a file within NetSuite.</p> <div data-bbox="407 1339 1370 1482">  Note: This object only encapsulates a file's metadata. Content is only loaded into memory (and returned as a string) when you call the File.getContents(). Content from CSV or text files can be accessed line by line using File.appendLine(options) or File.lines.iterator(). </div> <div data-bbox="407 1499 1370 1558">  Important: Binary content must be base64 encoded. </div> <p>Create a new <code>file.File</code> Object (up to 10MB in size) with the file.create(options) method. After you create a new <code>file.File</code>, you can:</p> <ul style="list-style-type: none"> ■ upload it to the NetSuite file cabinet with the File.save() method. ■ attach it to an email or fax without saving it to the file cabinet. <div data-bbox="407 1738 1370 1852">  Important: If you want to save the file to the NetSuite file cabinet, you must set a NetSuite file cabinet folder with the File.folder property. You must do this before you call File.save(). </div> <p>For a complete list of this object's methods and properties, see File Object Members.</p>
---------------------------	--

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
    name: 'test.txt',
    fileType: file.Type.PLAINTEXT,
    contents: 'Hello World\nHello World'
});
fileObj.folder = 30;
var fileId = fileObj.save();
...

//Add additional code
```

File.getContents()

Method Description	Method used to return the content of the file. <div> Important: Content held in memory is limited to 10MB.</div> <div> Note: You can access CSV or text files (including files over 10MB) using <code>File.appendLine(options)</code> or <code>File.lines.iterator()</code>.</div>
Returns	The file content as a string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_FILE_CONTENT_SIZE_EXCEEDED	The file content you are attempting to access exceeds the maximum allowed size of 10 MB.	You attempt to return the content of a file larger than 10MB.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 145
});
if (fileObj.size < 10485760){
    fileObj.getContents();
}
...
//Add additional code
```

File.appendLine(options)

Method Description	Method used to insert a line to the end of a file. This method can be used on text or .csv files. <div> Important: Content held in memory is limited to 10MB. Therefore, each line must be less than 10MB. </div>
Returns	file.File Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/file Module
Since	2017.1

Parameters



Note: The options parameter is a JavaScript object. The table below describes the name:value pairs that make up the object.

Parameter	Type	Required / Optional	Description	Since
options.value	string	required	Object containing a string to insert at the end of the file.	2017.1

Errors

Error Code	Message	Thrown If
SSS_FILE_CONTENT_SIZE_EXCEEDED	The content you are attempting to access exceeds the maximum	You attempt to return the content of a line larger than 10MB.

Error Code	Message	Thrown If
	allowed size of 10 MB.	
YOU_CANNOT_WRITE_TO_A_FILE_AFTER_YOU_BEGAN_READING_FROM_IT		You call File.appendLine(options) after calling File.lines.iterator() . To avoid receiving the error, call File.resetStream() or save the file.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 145
});
fileObj.appendLine({
    value: 'hello world'
});
...
//Add additional code
```

File.lines.iterator()

Method Description	<p>Method used to pass the next line as an argument to a developer-defined function. You can call this method multiple times to loop over the file contents as a stream. Return <code>false</code> to stop the loop. Return <code>true</code> to continue the loop. By default, <code>false</code> is returned when the end of the file is reached. This method can be used on text or .csv files.</p> <div> Important: Content held in memory is limited to 10MB. Therefore, each line must be less than 10MB. </div>
Returns	Boolean <code>true</code> <code>false</code>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/file Module
Since	2017.1

Parameters

Parameter	Type	Required / Optional	Description	Since
lineContext	iterator	required	Iterator which provides the next line of text from the text file to the iterator function.	2017.1

Errors

Error Code	Message	Thrown If
SSS_FILE_CONTENT_SIZE_EXCEEDED	The content you are attempting to access exceeds the maximum allowed size of 10 MB.	You attempt to return the content of a line larger than 10MB.
YOU_CANNOT_READ_FROM_A_FILE_AFTER_YOU_BEGAN_WRITING_TO_IT		You call <code>File.lines.iterator()</code> after calling <code>File.appendLine(options)</code> . Call <code>File.resetStream()</code> or save the file.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var iterator = invoiceFile.lines.iterator();

//Skip the first line (CSV header)
iterator.each(function () {return false;});
iterator.each(function (line)
{
    // This function updates the total by
    // adding the amount on each line to it
    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;
});
...
//Add additional code
```

File.resetStream()

Method Description	<p>Method used to reset the file contents. Serves as an undo action on any unsaved content written with <code>File.appendLine(options)</code> or <code>File.lines.iterator()</code>. Use this method to reset the reading and writing streams that may have been opened by <code>File.appendLine(options)</code> or <code>File.lines.iterator()</code>. The line pointer (or read iterator) is also set to its previous state. This method can be used on text or .csv files.</p> <div> Important: To use this method, each line must be less than 10MB. </div>
Returns	Void

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/file Module
Since	2017.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var afile = file.create({
    name: 'tmp3.txt',
    fileType: 'PLAINTEXT',
    contents:'one line'
});
afile.appendLine({
    value:'line two'
});
afile.resetStream();
afile.lines(function f({});
...

//Add additional code
```

File.save()

Method Description	<p>Method used to:</p> <ul style="list-style-type: none"> Upload a new file to the NetSuite file cabinet. Save an updated file to the NetSuite file cabinet. <div> Note: The <code>File.save()</code> method streams files of any size, provided that the file to save or upload meets file cabinet limits. </div> <div> Important: If you want to save the file to the NetSuite file cabinet, you must set a NetSuite file cabinet folder with the <code>File.folder</code> property. You must do this before you call <code>File.save()</code>. </div>
Returns	The internal ID of the file as a number.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	20 usage units
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
INVALID_KEY_OR_REF	Invalid folder reference key <passed folder ID>.	The File.folder property is set to an invalid folder ID.
SSS_MISSING_REQD_ARGUMENT	Please enter value(s) for: Folder	The File.folder property is not set before <code>save()</code> is called.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
    name      : 'test.txt',
    fileType: file.Type.PLAINTEXT,
    contents: 'Hello World\nHello World'
});
fileObj.folder = 30;
var fileId = fileObj.save();
...
//Add additional code
```

File.description

Property Description	The description of a file. In the UI, the value of <code>description</code> displays in the Description field on the file record.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
fileObj.description = 'my test file';
var fileId = fileObj.save();
...

```

```
//Add additional code
```

File.encoding

Property Description	The character encoding on a file. Value is set with the file.Encoding enum.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
  name      : 'test.txt',
  fileType: file.Type.PLAINTEXT,
  contents: 'Hello World\nHello World'
});
fileObj.encoding = file.Encoding.MAC_ROMAN;
fileObj.folder = 30;
var fileId = fileObj.save();
...

//Add additional code
```

File.fileType

Property Description	The file type of a file. This property is read-only. You must set the file type by passing in a file.Type enum value to file.create(options) .
Type	enum
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property after it is set with file.create(options) .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
  id: 145
});
log.debug({
  details: "File Type: " + fileObj.fileType
});
...

//Add additional code
```

File.folder

Property Description	The internal ID of a file's folder within the NetSuite file cabinet. Before you upload a file to the NetSuite file cabinet with File.save() , you must set its file cabinet folder with the <code>folder</code> property.
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
  name: 'test.txt',
  fileType: file.Type.PLAINTEXT,
  contents: 'Hello World\nHello World'
});
fileObj.folder = 30;
var fileId = fileObj.save();
...

//Add additional code
```

File.id

Property Description	The internal ID of the file within the NetSuite file cabinet. This value is automatically generated by NetSuite.
-----------------------------	--

	This property is read-only.
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
  id: 'Images/myImageFile.jpg'
});
log.debug({
  details: "File ID: " + fileObj.id
});
...
//Add additional code
```

File.isInactive

Property Description	The inactive status of a file. If set to true, the file is inactive. The default value is <code>false</code> . When a file is inactive, it does not display in the UI unless you select Show Inactives on the File Cabinet page.
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
```

```

...
var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
fileObj.name = 'myOldImageFile.jpg';
fileObj.isInactive = true;
var fileId = fileObj.save();
...
//Add additional code

```

File.isOnline

Property Description	<p>The Available without Login status of a file. If set to <code>true</code>, users can download the file outside of a current NetSuite login session. The default value is <code>false</code>.</p> <div>  Important: This property holds the value of the Available without Login setting found on the file record. It does not reflect the value of the Available Without Login setting found on the Suitelet script deployment record. </div> <p>The Available without Login setting is primarily used for SuiteCommerce websites. When this setting is enabled, websites can access media files in the NetSuite file cabinet without a current NetSuite login session.</p>
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```

//Add additional code
...
var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
fileObj.isOnline = true;
var fileId = fileObj.save();
...
//Add additional code

```

File.isText

Property Description	<p>Indicates whether a file type is text-based. This property is read-only.</p>
Type	boolean <code>true</code> <code>false</code>

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 145
});
if (fileObj.isText === true){
    ...
}
...
//Add additional code
```

File.name

Property Description	The name of a file.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
fileObj.name = 'myOldImageFile.jpg';
var fileId = fileObj.save();
```

```
...
//Add additional code
```

File.path

Property Description	<p>The relative path to a file in the NetSuite file cabinet.</p> <div>  Note: If the folder is not set with the <code>file.create(options)</code> method, this property holds the file name until the <code>File.folder</code> property is defined. </div> <p>This property is read-only.</p>
Type	string
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 145
});
log.debug({
    details: "File Path: " + fileObj.path
});
...
//Add additional code
```

File.size

Property Description	<p>The size of a file in bytes.</p> <p>This property is read-only.</p> <div>  Note: You can use this value to determine if the file is within size limits for <code>File.getContents()</code>. Size will reflect any lines you have streamed into a file. For example, the original file size plus lines appended. </div>
Type	number

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
log.debug({
    details: "File Size: " + fileObj.size
});
...
//Add additional code
```

File.url

Property Description	The URL of a file. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/file Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code ...
```

```

var fileObj = file.load({
    id: 'Images/myImageFile.jpg'
});
log.debug({
    details: "File URL: " + fileObj.url
});
...
//Add additional code

```

file.create(options)

Method Description	Method used to create a new file in the NetSuite file cabinet.  Important: Content held in memory is limited to 10MB.
Returns	file.File
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/file Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.name</code>	string	required	<ul style="list-style-type: none"> The file name and extension. Sets the value for the File.name property. 	2015.2
<code>options.fileType</code>	enum	required	<ul style="list-style-type: none"> The file type. Sets the value for the File.fileType property. This property is read-only and cannot be changed after the file is created. Use the file.Type enum to set the value. 	2015.2
<code>options.contents</code>	string	optional	<ul style="list-style-type: none"> The file content. File content is lazy loaded; there is no property for it. If the file type is binary (for example, PDF), the file content must be base64 encoded. 	2015.2
<code>options.description</code>	string	optional	<ul style="list-style-type: none"> The file description. In the UI, the value of description displays the Description field on the file record. Sets the value for the File.description property. 	2016.2
<code>options.folder</code>	number	optional	<ul style="list-style-type: none"> The internal ID of the folder within the NetSuite file cabinet. You must set the file cabinet folder 	2016.1

Parameter	Type	Required / Optional	Description	Since
			before you upload a file to the NetSuite file cabinet with <code>File.save()</code> . <ul style="list-style-type: none"> ■ Sets the value for the <code>File.folder</code> property. 	
<code>options.encoding</code>	string	optional	<ul style="list-style-type: none"> ■ The character encoding on a file. ■ Sets the value for the <code>File.encoding</code> property. ■ Use the <code>file.Encoding</code> enum to set the value. 	2016.2
<code>options.isInactive</code>	boolean false true	optional	<ul style="list-style-type: none"> ■ The inactive status of a file. If set to true, the file is inactive. The default value is false. When a file is inactive, it does not display in the UI unless you select Show Inactives on the File Cabinet page. ■ Sets the value for the <code>File.isInactive</code> property. 	2016.2
<code>options.isOnline</code>	boolean false true	optional	<ul style="list-style-type: none"> ■ The Available without Login status of a file. If set to true, users can download the file outside of a current netSuite login session. The default value is false. ■ Sets the value for the <code>File.isOnline</code> property. 	2016.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	<name of missing parameter>	A required argument is not passed.
SSS_INVALID_TYPE_ARG	You have entered an invalid type argument: <passed type argument>	The argument for <code>File.fileType</code> is invalid.
SSS_FILE_CONTENT_SIZE_EXCEEDED	The file you are trying to create exceeds the maximum allowed file size of 10.0 MB.	You attempt to create a file larger than 10MB.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
    name: 'test.txt',
    fileType: file.Type.PLAINTEXT,
    contents: 'Hello World\nHello World',
    description: 'This is a plain text file.',
    encoding: file.Encoding.UTF8,
    folder: 30,
    isOnline: true
});
...
//Add additional code
```

file.delete(options)

Method Description	Method used to delete an existing file from the NetSuite file cabinet.
Returns	The internal ID of the deleted file
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	20 usage units
Module	N/file Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	number string	required	<ul style="list-style-type: none"> Internal ID of the file. To find the internal ID of the file in the UI, click Documents > Files > File Cabinet. 	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	<name of missing parameter>	A required argument is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
  name: 'test.txt',
  fileType: file.Type.PLAINTEXT,
  contents: 'Hello World\nHello World'
});
fileObj.folder = 30;
var fileId = fileObj.save();

file.delete({
  id: fileId
});
...
//Add additional code
```

file.load(options)

Method Description	Method used to load an existing file from the NetSuite file cabinet.
Returns	An existing file.File .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 usage units
Module	N/file Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	number string	required	<ul style="list-style-type: none"> Pass one of the following: <ul style="list-style-type: none"> Internal ID of the file as a number or a string. The relative file path to the file in the file cabinet. For example, <code>'Images/myImageFile.jpg'</code>. To find the internal ID of the file in the UI, select Documents > Files > File Cabinet. 	2015.2

Errors

Error Code	Message	Thrown If
INSUFFICIENT_PERMISSION	You do not have access to the media item you selected.	Internal ID passed is invalid.
RCRD_DSNT_EXIST	That record does not exist. path: {path}	Relative file path passed is invalid.
SSS_MISSING_REQD_ARGUMENT	<name of missing parameter>	A required argument is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.load({
  id: 'Images/myImageFile.jpg'
});
fileObj.description = 'my test file';
var fileId = fileObj.save();
...
```

```
//Add additional code
```

file.Encoding

Enum Description	<p>Enumeration that holds the string values for supported character encoding. This enum is used to set the value of the File.encoding property.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/file Module
Since	2015.2

Values

Value	Character Set
UTF_8	Unicode
WINDOWS_1252	Western
ISO_8859_1	Western
GB18030	Chinese Simplified
SHIFT_JIS	Japanese
MAC_ROMAN	Western
GB2312	Chinese Simplified
BIG5	Chinese Traditional

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
  name: 'test.txt',
  fileType: file.Type.PLAINTEXT,
  contents: 'Hello World\nHello World'
});
fileObj.encoding = file.Encoding.MAC_ROMAN;
fileObj.folder = 30;
var fileId = fileObj.save();
...
```



```
//Add additional code
```

file.Type

Enum Description	<p>Enumeration that holds the string values for supported file types. This enum is used to set the value of the File.fileType property.</p> <p>Note that the File.fileType property is read only. It's value must be set with file.create(options). See N/file Module Script Sample for an example.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/file Module
Since	2015.2

Values

<ul style="list-style-type: none"> ■ APPCACHE ■ AUTOCAD ■ BMPIMAGE ■ CERTIFICATE ■ CONFIG ■ CSV ■ EXCEL ■ FLASH ■ FREEMARKER ■ GIFIMAGE ■ GZIP ■ HTMLDOC ■ ICON ■ JAVASCRIPT ■ JPGIMAGE 	<ul style="list-style-type: none"> ■ JSON ■ MESSENGERFC ■ MP3 ■ MPEGMOVIE ■ MSPROJECT ■ PDF ■ PJPGIMAGE ■ PLAINTEXT ■ PNGIMAGE ■ POSTSCRIPT ■ POWERPOINT ■ QUICKTIME 	<ul style="list-style-type: none"> ■ RTF ■ SCSS ■ SMS ■ STYLESHEET ■ SVG ■ TAR ■ TIFFIMAGE ■ VISIO ■ WEBAPPPAGE ■ WEBAPPScript ■ WORD ■ XMLDOC ■ XSD ■ ZIP
--	--	--

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/file Module Script Sample](#).

```
//Add additional code
...
var fileObj = file.create({
  name    : 'test.txt',
```

```

fileType: file.Type.PLAINTEXT,
contents: 'Hello World\nHello World'
});
fileObj.folder = 30;
var fileId = fileObj.save();
...
//Add additional code

```

N/format Module

Use the format module to parse formatted data into strings and to convert strings into a specified format. The format module formats data according to personal preferences set on the Set Preferences page, accessible from Home > Set Preferences . See the help topic [Setting Personal Preferences](#).

- [N/format Module Members](#)
- [N/format Module Script Samples](#)

N/format Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>format.format(options)</code>	string Date	Client and server-side scripts	Takes a raw value and returns a formatted value. <div>  Note: This method is overloaded when you format a <code>datetime</code> or <code>datetimez</code> value. </div>
	<code>format.parse(options)</code>	Date string number	Client and server-side scripts	Takes a formatted value and returns a raw value. <div>  Note: This method is overloaded when you format a <code>datetime</code> or <code>datetimez</code> value. </div>
Enum	<code>format.Type</code>	enum	Client and server-side scripts	Holds the string values for the supported field types. This enum is used to set the value of the <code>options.type</code> parameter.
	<code>format.Timezone</code>	enum	Client and server-side scripts	Holds the string values for supported time zone formats. This enum is used to set the value of the <code>options.timezone</code> parameter.

N/format Module Script Samples

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Example 1

The following example parses a string (formatted according to the user preference) to a raw Date Object, and then parses it back to the formatted string. This example uses `format.parse(options)` and `format.format(options)`.

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

```
/**
 *@NApiVersion 2.x
 */
require(['N/format'],
  function(format) {
    function parseAndFormatDateString() {
      // Assume 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
      });
    }
    parseAndFormatDateString();
    // "07/28/2015"
  });
```

Example 2

The following example parses a string (formatted according to the user 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 debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 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
```

```
});
```

Example 3

The following example formats a raw number value (formatted according to the user preference) to a string, using `format.format(options)`.

```
/**
 *@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
  });
```

Example 4

The following example formats the time of day to a string, using `format.format(options)`.

 **Note:** This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 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
  });
```

format.format(options)

Method Description	Formats a value from the raw value to its appropriate preference format. <div> Note: This method is overloaded when you format a <code>datetime</code> or <code>datetimez</code> value.</div>
Returns	The formatted value as a string. If a <code>datetime</code> or <code>datetimez</code> value is specified, the <code>Date</code> Object is returned in the user's local app time zone.

	<p>Note: If an invalid value is given, the original value passed to <code>options.value</code> is returned.</p> <p>Note: For client side scripts, the string returned is based on the user's system time. For server-side scripts, the string returned is based on the system time of the server your NetSuite system is running on.</p>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types
Governance	None
Module	N/format Module
Since	2015.2

Parameters

This method is overloaded when you format a `datetime` or `datetimez` value.

Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	Date string number	required	The input data to format.	2015.2
<code>options.type</code>	string	required	The field type (for example, <code>DATE</code> , <code>CURRENCY</code> , <code>INTEGER</code>). Set using the format.Type enum.	2015.2

The table below applies to `datetime` and `datetimez` values only.

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	Date	required	The Date Object being converted into a string	2015.2
<code>options.type</code>	string	required	The field type (either <code>DATETIME</code> or <code>DATETIMEZ</code>). Set using the format.Type enum.	2015.2
<code>options.timezone</code>	enum number	optional	The time zone specified for the returned string. Set using the format.Timezone enum or key. If a time zone is not specified, the time zone is set based on user preference. If the time zone is invalid, the time zone is set to GMT.	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/format Module Script Samples .
--

```
// Add additional code
...
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
  ...
  // Add additional code

```

format.parse(options)

Method Description	<p>Parses a value from the appropriate preference format to its raw value. The appropriate preference format is the one selected in the Date Format field at Home > Set Preferences. For a <code>datetime</code> or <code>datetimetz</code> value, use this method to convert a Date Object into a string based on the specified timezone.</p> <div>  Note: This method is overloaded when you format a <code>datetime</code> or <code>datetimetz</code> value. </div>
Returns	<p>The parsed value as a Date string number <code>Datetime</code> or <code>datetimetz</code> values are returned as a string.</p> <div>  Note: If the value given is not valid or parsable, the original value passed to <code>options.value</code> is returned. </div>
Supported Script Types	<p>Client and server-side scripts For more information, see SuiteScript 2.0 Script Types</p>
Governance	None
Module	N/format Module
Since	2015.2

Parameters

This method is overloaded when you format a `datetime` or `datetimetz` value.

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	string	required	The input data to parse.	2015.2
<code>options.type</code>	string	required	The field type (for example, <code>DATE</code> , <code>CURRENCY</code> , <code>INTEGER</code>). Set using the <code>format.Type</code> enum.	2015.2

The table below applies to `datetime` and `datetimetz` values only.

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	string	required	The string that contains the date and time information in the specified timezone.	2015.2
<code>options.type</code>	string	required	The field type (either <code>DATETIME</code> or <code>DATETIMETZ</code>).	2015.2

Parameter	Type	Required / Optional	Description	Since
			Set using the format.Type enum.	
<code>options.timezone</code>	enum	optional	The time zone represented by the <code>options.value</code> string. Set using the format.Timezone enum. If a time zone is not specified, the time zone is based on user preference. If the time zone is invalid, the time zone is set to GMT.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/format Module Script Samples](#).

```
// Add additional code
...
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
    ...
    // Add additional code
```

format.Type

Enum Description	Enumeration that holds the string values for the supported field types. This enum is used to set the value of the <code>options.type</code> parameter when calling <code>format.format(options)</code> or <code>format.parse(options)</code> . <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types
Module	N/format Module
Since	2015.2

Values

<ul style="list-style-type: none"> ADDRESS CCEXPDATE CCNUMBER CCVALIDFROM CHECKBOX 	<ul style="list-style-type: none"> EMAIL EMAILS FLOAT FULLPHONE FUNCTION 	<ul style="list-style-type: none"> POSCURRENCY POSFLOAT POSINTEGER QUOTEDFUNCTION RADIO
---	---	--

<ul style="list-style-type: none"> ■ COLOR ■ CURRENCY ■ CURRENCY2 ■ DATE ■ DATETIME ■ DATETIMETZ 	<ul style="list-style-type: none"> ■ FURIGANA ■ IDENTIFIER ■ IDENTIFIERANYCASE ■ INTEGER ■ MMYDATE ■ NONNEGCURRENCY ■ NONNEGFLOAT ■ PACKAGE ■ PERCENT ■ PHONE 	<ul style="list-style-type: none"> ■ RATE ■ RATEHIGHPRECISION ■ TEXT ■ TIME ■ TIMEOFDAY ■ TIMETRACK ■ URL
--	---	--

Syntax

Note: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/format Module Script Samples](#).

```
// Add additional code
...
function formatTimeOfDay() {
    // Assume the time format is hh:mm (24 hours)
    var now = new Date(); // Say it's 7:01PM right now.
    var formattedTime = format.format({value: now, type: format.Type.TIMEOFDAY})
    });
...
// Add additional code
```

format.Timezone

Enum Description	<p>Enumeration that holds the string values for supported time zone formats. This enum is used to set the value of the <code>options.timezone</code> parameter when calling <code>format.format(options)</code> or <code>format.parse(options)</code>.</p> <p>Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</p>
Supported Script Types	<p>Client and server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types</p>
Module	N/format Module
Since	2015.2

Values

This table defines all valid time zone names in Olson Value format and includes daylight savings time rules for each time zone. Olson Values are maintained by the International Assigned Numbers Authority (IANA) in an international standard time zone database. The values that populate the Time Zone dropdown list found at Home > Set Preferences are also based on these values.

When working with alternate time zones in SuiteScript, use these enumeration values. If necessary, you can use the numerical key in place of an Olson Value string. For example, to source a custom timezone dropdown list.

Key	Olson Value	Description
1	ETC_GMT_PLUS_12: 'Etc/GMT+12'	(GMT-12:00) International Date Line West
2	PACIFIC_SAMOA: 'Pacific/Samoa'	(GMT-11:00) Midway Island, Samoa
3	PACIFIC_HONOLULU: 'Pacific/Honolulu'	(GMT-10:00) Hawaii
4	AMERICA_ANCHORAGE: 'America/Anchorage'	(GMT-09:00) Alaska
5	AMERICA_LOS_ANGELES: 'America/Los_Angeles'	(GMT-08:00) Pacific Time (US & Canada)
6	AMERICA_TIJUANA: 'America/Tijuana'	(GMT-08:00) Tijuana, Baja California
7	AMERICA_DENVER: 'America/Denver'	(GMT-07:00) Mountain Time (US & Canada)
8	AMERICA_PHOENIX: 'America/Phoenix'	(GMT-07:00) Arizona
9	AMERICA_CHIHUAHUA: 'America/Chihuahua'	(GMT-07:00) Chihuahua, La Paz, Mazatlan - New
10	AMERICA_CHICAGO: 'America/Chicago'	(GMT-06:00) Central Time (US & Canada)
11	AMERICA_REGINA: 'America/Regina'	(GMT-06:00) Saskatchewan
12	AMERICA_GUATEMALA: 'America/Guatemala'	(GMT-06:00) Central America
13	AMERICA_MEXICO_CITY: 'America/Mexico_City'	(GMT-06:00) Guadalajara, Mexico City, Monterrey - Old
14	AMERICA_NEW_YORK: 'America/New_York'	(GMT-05:00) Eastern Time (US & Canada)
15	US_EAST_INDIANA: 'US/East-Indiana'	(GMT-05:00) Indiana (East)
16	AMERICA_BOGOTA: 'America/Bogota'	(GMT-05:00) Bogota, Lima, Quito
17	AMERICA_CARACAS: 'America/Caracas'	(GMT-04:30) Caracas
18	AMERICA_HALIFAX: 'America/Halifax'	(GMT-04:00) Atlantic Time (Canada)
19	AMERICA_LA_PAZ: 'America/La_Paz'	(GMT-04:00) Georgetown, La Paz, San Juan
20	AMERICA_MANAUS: 'America/Manaus'	(GMT-04:00) Manaus
21	AMERICA_SANTIAGO: 'America/Santiago'	(GMT-04:00) Santiago
22	AMERICA_ST_JOHNS: 'America/St_Johns'	(GMT-03:30) Newfoundland
23	AMERICA_SAO_PAULO: 'America/Sao_Paulo'	(GMT-03:00) Brasilia
24	AMERICA_BUENOS_AIRES: 'America/Buenos_Aires'	(GMT-03:00) Buenos Aires
25	ETC_GMT_PLUS_3: 'Etc/GMT+3'	(GMT-03:00) Cayenne
26	AMERICA_GODTHAB: 'America/Godthab'	(GMT-03:00) Greenland
27	AMERICA_MONTEVIDEO: 'America/Montevideo'	(GMT-03:00) Montevideo
28	AMERICA_NORONHA: 'America/Noronha'	(GMT-02:00) Mid-Atlantic
29	ETC_GMT_PLUS_1: 'Etc/GMT+1'	(GMT-01:00) Cape Verde Is.
30	ATLANTIC_AZORES: 'Atlantic/Azores'	(GMT-01:00) Azores
31	EUROPE_LONDON: 'Europe/London', GMT: 'GMT'	(GMT) Greenwich Mean Time : Dublin, Edinburgh, Lisbon, London
32	GMT: 'GMT'	(GMT) Casablanca

Key	Olson Value	Description
33	ATLANTIC_REYKJAVIK: 'Atlantic/Reykjavik'	(GMT) Monrovia, Reykjavik
34	EUROPE_WARSZAWA: 'Europe/Warsaw'	(GMT+01:00) Sarajevo, Skopje, Warsaw, Zagreb
35	EUROPE_PARIS: 'Europe/Paris'	(GMT+01:00) Brussels, Copenhagen, Madrid, Paris
36	ETC_GMT_MINUS_1: 'Etc/GMT-1'	(GMT+01:00) West Central Africa
37	EUROPE_AMSTERDAM: 'Europe/Amsterdam'	(GMT+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
38	EUROPE_BUDAPEST: 'Europe/Budapest'	(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
39	AFRICA_CAIRO: 'Africa/Cairo'	(GMT+02:00) Cairo
40	EUROPE_ISTANBUL: 'Europe/Istanbul'	(GMT+02:00) Athens, Bucharest, Istanbul
41	ASIA_JERUSALEM: 'Asia/Jerusalem'	(GMT+02:00) Jerusalem
42	ASIA_AMMAN: 'Asia/Amman'	(GMT+02:00) Amman
43	ASIA_BEIRUT: 'Asia/Beirut'	(GMT+02:00) Beirut
44	AFRICA_JOHANNESBURG: 'Africa/Johannesburg'	(GMT+02:00) Harare, Pretoria
45	EUROPE_KIEV: 'Europe/Kiev'	(GMT+02:00) Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
46	EUROPE_MINSK: 'Europe/Minsk'	(GMT+02:00) Minsk
47	AFRICA_WINDHOEK: 'Africa/Windhoek'	(GMT+02:00) Windhoek
48	ASIA_RIYADH: 'Asia/Riyadh'	(GMT+03:00) Kuwait, Riyadh
49	EUROPE_MOSCOW: 'Europe/Moscow'	(GMT+03:00) Moscow, St. Petersburg, Volgograd
50	ASIA_BAGHDAD: 'Asia/Baghdad'	(GMT+03:00) Baghdad
51	AFRICA_NAIROBI: 'Africa/Nairobi'	(GMT+03:00) Nairobi
52	ASIA_TEHRAN: 'Asia/Tehran'	(GMT+03:30) Tehran
53	ASIA_MUSCAT: 'Asia/Muscat'	(GMT+04:00) Abu Dhabi, Muscat
54	ASIA_BAKU: 'Asia/Baku'	(GMT+04:00) Baku
55	ASIA_YEREVAN: 'Asia/Yerevan'	(GMT+04:00) Caucasus Standard Time
56	ETC_GMT_MINUS_3: 'Etc/GMT-3'	(GMT+04:00) Tbilisi
57	ASIA_KABUL: 'Asia/Kabul'	(GMT+04:30) Kabul
58	ASIA_KARACHI: 'Asia/Karachi'	(GMT+05:00) Islamabad, Karachi
59	ASIA_YEKATERINBURG: 'Asia/Yekaterinburg'	(GMT+05:00) Ekaterinburg
60	ASIA_TASHKENT: 'Asia/Tashkent'	(GMT+05:00) Tashkent
61	ASIA_CALCUTTA: 'Asia/Calcutta'	(GMT+05:30) Chennai, Kolkata, Mumbai, New Delhi
62	ASIA_KATMANDU: 'Asia/Katmandu'	(GMT+05:45) Kathmandu
63	ASIA_ALMATY: 'Asia/Almaty'	(GMT+06:00) Novosibirsk
64	ASIA_DHAKA: 'Asia/Dhaka'	(GMT+06:00) Astana, Dhaka
65	ASIA_RANGOON: 'Asia/Rangoon'	(GMT+06:30) Yangon (Rangoon)

Key	Olson Value	Description
66	ASIA_BANGKOK: 'Asia/Bangkok'	(GMT+07:00) Bangkok, Hanoi, Jakarta
67	ASIA_KRASNOYARSK: 'Asia/Krasnoyarsk'	(GMT+07:00) Krasnoyarsk
68	ASIA_HONG_KONG: 'Asia/Hong_Kong'	(GMT+08:00) Beijing, Chongqing, Hong Kong, Urumqi
69	ASIA_KUALA_LUMPUR: 'Asia/Kuala_Lumpur'	(GMT+08:00) Kuala Lumpur, Singapore
70	ASIA_TAIPEI: 'Asia/Taipei'	(GMT+08:00) Taipei
71	AUSTRALIA_PERTH: 'Australia/Perth'	(GMT+08:00) Perth
72	ASIA_IRKUTSK: 'Asia/Irkutsk'	(GMT+08:00) Irkutsk
73	ASIA_MANILA: 'Asia/Manila'	(GMT+08:00) Manila
74	ASIA_SEOUL: 'Asia/Seoul'	(GMT+09:00) Seoul
75	ASIA_TOKYO: 'Asia/Tokyo'	(GMT+09:00) Osaka, Sapporo, Tokyo
76	ASIA_YAKUTSK: 'Asia/Yakutsk'	(GMT+09:00) Yakutsk
77	AUSTRALIA_DARWIN: 'Australia/Darwin'	(GMT+09:30) Darwin
78	AUSTRALIA_ADELAIDE: 'Australia/Adelaide'	(GMT+09:30) Adelaide
79	AUSTRALIA_SYDNEY: 'Australia/Sydney'	(GMT+10:00) Canberra, Melbourne, Sydney
80	AUSTRALIA_BRISBANE: 'Australia/Brisbane'	(GMT+10:00) Brisbane
81	AUSTRALIA_HOBART: 'Australia/Hobart'	(GMT+10:00) Hobart
82	PACIFIC_GUAM: 'Pacific/Guam'	(GMT+10:00) Guam, Port Moresby
83	ASIA_VLADIVOSTOK: 'Asia/Vladivostok'	(GMT+10:00) Vladivostok
84	ASIA_MAGADAN: 'Asia/Magadan'	(GMT+11:00) Magadan, Solomon Is., New Caledonia
85	PACIFIC_KWAJALEIN: 'Pacific/Kwajalein'	(GMT+12:00) Fiji, Marshall Is.
86	PACIFIC_AUCKLAND: 'Pacific/Auckland'	(GMT+12:00) Auckland, Wellington
87	PACIFIC_TONGATAPU: 'Pacific/Tongatapu'	(GMT+13:00) Nuku'alofa

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/format Module Script Samples](#).

```
// Add additional code
...
var date = new Date(); //Mon Aug 24 2015 17:27:16 GMT-0700 (Pacific Daylight Time)
var TOKYO = format.format({
    value: date,
    type: format.Type.DATETIME,
    timezone: format.Timezone.ASIA_TOKYO
}); //Returns "8/25/2015 9:27:16 am"

var NEWYORK = format.format({
    value: date,
    type: format.Type.DATETIME,
```

```

        timezone: format.Timezone.AMERICA_NEW_YORK
    }); //Returns "8/24/2015 8:27:16 pm"

var dateStr = "03/17/2015 09:00:00 pm"
var TOKYO_2 = format.parse({
    value: dateStr,
    type: format.Type.DATETIME,
    timezone: format.Timezone.ASIA_TOKYO
}); //Returns Date object [[ Tue Mar 17 2015 05:00:00 GMT-0700 (PDT) ]]

var NEWYORK_2 = format.parse({
    value: dateStr,
    type: format.Type.DATETIME,
    timezone: format.Timezone.AMERICA_NEW_YORK
}); //Returns Date object [[ Tue Mar 17 2015 18:00:00 GMT-0700 (PDT) ]]
...
// Add additional code

```

N/http Module

Use the http module to make HTTP calls from server-side or client-side scripts. On the client-side, this module also provides the ability to make cross-domain HTTP requests using NetSuite servers as a proxies.

All HTTP content types are supported.

Note: The http module does not accept the HTTPS protocol. Use the [N/https Module](#) for that purpose.

- [N/http Module Members](#)
- [ClientResponse Object Members](#)
- [ServerRequest Object Members](#)
- [ServerResponse Object Members](#)
- [N/http Module Script Sample](#)

General HTTP Header Blacklist

Be aware that certain headers cannot be set manually when using http module methods. If a script attempts to set values for any of the following headers, the values are discarded. These headers are described in the following table.

■ Connection	■ Transfer-Encoding
■ Content-Length	■ Upgrade
■ Host	■ Via
■ Trailer	

Suitelet Response HTTP Header Blacklist

In addition to the headers described in [General HTTP Header Blacklist](#), certain headers cannot be set manually when interacting with the [http.ServerResponse](#) objects sent by Suitelets. If a script attempts to set values for any of these headers, the system throws an SSS_INVALID_HEADER error. These headers are described in the following table.

<ul style="list-style-type: none"> Access-Control-Allow-Origin Allow Connection Content-Length Content-Location Content-MD5 Content-Range 	<ul style="list-style-type: none"> Date Location Proxy-Authenticate Retry-After Server Trailer Via 	<ul style="list-style-type: none"> Warning WWW-Authenticate
--	---	---

N/http Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	http.ClientResponse	read-only Object	Server-side scripts	Encapsulates the response to an HTTP client request.
	http.ServerRequest	read-only Object	Server-side scripts	Encapsulates the HTTP request information sent to an HTTP server. For example, a request received by a Suitelet or RESTlet.
	http.ServerResponse	Object	Server-side scripts	Encapsulates the response from an HTTP server to an HTTP request. For example, a response from a Suitelet or RESTlet.
Method	http.delete(options)	http.ClientResponse	Server-side scripts	Sends an HTTP DELETE request and returns the response.
	http.delete.promise(options)	http.ClientResponse	Client scripts	Sends an HTTP DELETE request asynchronously and returns the response.
	http.get(options)	http.ClientResponse	Server-side scripts	Sends an HTTP GET request and returns the response.
	http.get.promise(options)	http.ClientResponse	Client scripts	Sends an HTTP GET request asynchronously and returns the response.
	http.post(options)	http.ClientResponse	Server-side scripts	Sends an HTTP POST request and returns the response.
	http.post.promise(options)	http.ClientResponse	Client scripts	Sends an HTTP POST request asynchronously and returns the response.
	http.put(options)	http.ClientResponse	Server-side scripts	Sends an HTTP PUT request and returns the response.
	http.put.promise(options)	http.ClientResponse	Client scripts	Sends an HTTP PUT request asynchronously and returns the response.
	http.request(options)	http.ClientResponse	Server-side scripts	Sends an HTTP request and returns the response.
	http.request.promise(options)	http.ClientResponse	Client scripts	Sends an HTTP request asynchronously and returns the response.
Enum	http.CacheDuration	enum	Server-side scripts	Holds the string values for supported cache

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				durations. This enum is used to set the value of the ServerResponse.setCdnCacheable(options) property.
	http.Method	enum	Server-side scripts	Holds the string values for supported HTTP requests. This enum is used to set the value of http.request(options) and ServerRequest.method .
	http.RedirectType	enum	Server-side scripts	Holds the string values for supported NetSuite resources that you can redirect to. This enum is used to set the value of the <code>type</code> argument for ServerResponse.sendRedirect(options) .

ClientResponse Object Members

The following members are called on [http.ClientResponse](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	ClientResponse.body	read-only string	Server-side scripts	The client response body.
	ClientResponse.code	read-only number	Server-side scripts	The client response code.
	ClientResponse.headers	read-only Object	Server-side scripts	The client response headers.

ServerRequest Object Members

The following members are called on the [http.ServerRequest](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	ServerRequest.getLineCount(options)	number	Server-side scripts	Returns the number of lines in a sublist.
	ServerRequest.getSublistValue(options)	string	Server-side scripts	Returns the value of a sublist line item.
Property	ServerRequest.body	read-only string	Server-side scripts	The server request body.
	ServerRequest.files	read-only Object	Server-side scripts	The server request files.
	ServerRequest.headers	read-only Object	Server-side scripts	The server request headers.
	ServerRequest.method	http.Method enum	Server-side scripts	The HTTP method for the server request.
	ServerRequest.parameters	read-only Object	Server-side scripts	The server request parameters.
	ServerRequest.url	read-only string	Server-side scripts	The server request URL.

ServerResponse Object Members

The following members are called on the [http.ServerResponse](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	ServerResponse.addHeader(options)	void	Server-side scripts	Adds a header to the response.
	ServerResponse.getHeader(options)	void	Server-side scripts	Returns the value of a response header.
	ServerResponse.renderPdf(options)	void	Server-side scripts	Generates and renders a PDF directly to the response.
	ServerResponse.sendRedirect(options)	void	Server-side scripts	Sets the redirect URL by resolving to a NetSuite resource.
	ServerResponse.setCdnCacheable(options)	void	Server-side scripts	Sets CDN caching for a period of time.
	ServerResponse.setHeader(options)	void	Server-side scripts	Sets the value of a response header.
	ServerResponse.write(options)	void	Server-side scripts	Writes information (text/xml/html) to the response.
	ServerResponse.writeFile(options)	void	Server-side scripts	Writes a file to the response.
	ServerResponse.writeLine(options)	void	Server-side scripts	Writes line information (text/xml/html) to the response.
	ServerResponse.writePage(options)	void	Server-side scripts	Generates a page.
Property	ServerResponse.headers	Object	Server-side scripts	The server response headers.

N/http Module Script Sample

Example 1

The following example shows an HTTP GET request for a URL.

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 */
require(['N/http'],
function(http) {
    function sendGetRequest() {
        var response = http.get({
            url: 'http://www.google.com'
```

```

    });
  }
  sendGetRequest();
});

```

Example 2

The following example is designed to redirect to new sales order record, and will set entity to 6. (Assuming there is an entity with number 6, if there's not, then entity will remain blank.)

Note: This sample script uses the define function. Note that you cannot use *Ad Hoc Debugging* to step through a define function. You must use *Deployed Debugging* to step through this script.

```

/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define([ 'N/record', 'N/http' ],
function(record, http) {
    function onRequest(context) {
        context.response.sendRedirect({
            type: http.RedirectType.RECORD,
            identifier: record.Type.SALES_ORDER,
            parameters: {
                entity : 6
            }
        });
    }

    return {
        onRequest : onRequest
    };
});

```

http.ClientResponse

Object Description	Encapsulates the response to an HTTP client request. This object is read-only. For a complete list of this object's properties, see ClientResponse Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
```



```

...
var clientResponse = http.get({
    url: 'http://www.google.com'
});
...
//Add additional code

```

ClientResponse.body

Property Description	The client response body. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ClientResponse
Sibling Object Members	ClientResponse Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```

//Add additional code
...
var response = http.get({
    url: 'http://www.google.com'
});
log.debug({
    title: 'Client Response Body',
    details: http.response.body
});
...
//Add additional code

```

ClientResponse.code

Property Description	The client response code. This property is read-only.
Type	number

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ClientResponse
Sibling Object Members	ClientResponse Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = http.get({
    url: 'http://www.google.com'
});
log.debug({
    title: 'Client Response Code',
    details: http.response.code
});
...
//Add additional code
```

ClientResponse.headers

Property Description	The response header or headers. This property is read-only.
Type	object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ClientResponse
Sibling Object Members	ClientResponse Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = http.get({
    url: 'http://www.google.com'
});
log.debug({
    title: 'Client Response Header',
    details: http.response.headers
});
...
//Add additional code
```

http.ServerRequest

Object Description	Encapsulates the HTTP request information to an HTTP server. For example, a request received by a Suitelet or RESTlet. This object is read-only. For a complete list of this object's methods and properties, see ServerRequest Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverRequest.getLineCount({
    group: 'sublistId'
});
...
//Add additional code
```

ServerRequest.getLineCount(options)

Method Description	Method used to return the number of lines in a sublist.
Returns	The number of lines in a sublist as a number.

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
<code>options.group</code>	string	required	The sublist internal ID.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverRequest.getLineCount({
    group: 'sublistId'
});
...
//Add additional code
```

ServerRequest.getSublistValue(options)

Method Description	Method used to return the value of a sublist line item.
Returns	The value of the sublist line item as a string.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members

Since	2015.2
-------	--------

Parameters

Parameter	Type	Required / Optional	Description	Since
options.group	string	required	The sublist internal ID.	2015.2
options.name	string	required	The sublist line item ID (name of the field).	2015.2
options.line	string	required	The sublist line number.	2015.2



Note: Sublist index starts at 0.

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverRequest.getSublistValue({
    group: 'item',
    name: 'amount',
    line: '2'
});
...
//Add additional code
```

ServerRequest.body

Property Description	The server request body. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Body',
  details: http.request.body
});
...
//Add additional code
```

ServerRequest.files

Property Description	The server request files. This property is read-only.
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippets show the syntax for this member. They are not functional examples. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Files',
  details: http.request.files
});
```

```
...
//Add additional code
```

```
var file = request.files['file_id'];
```

ServerRequest.headers

Property Description	<p>This object represents a series of key/value pairs. Each pair represents a server request header name and its value.</p> <p>Typically, this object encapsulates two iterations of each header name: one in lower case and another in title case. This behavior is designed so that you can use either lower case or title case when you reference a header. However, the existence of title-case iterations of header names is not guaranteed. For best results, refer to header names using all lower-case letters (and hyphens, when applicable).</p> <p>This property is read-only.</p> <div>  Important: The server request headers and their values are subject to change. If you use these headers in your scripts, you are responsible for testing them to make sure that they contain the information you need. For example, when making an HTTP call to a Suitelet, some headers might be filtered out. Filtering can occur if the headers affect how NetSuite processes the request internally. These filtered headers are not available to the Suitelet, so you should test to see whether a header was filtered out. If so, use a different header instead. </div>
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Headers',
  details: http.request.headers
```

```
});
...
//Add additional code
```

ServerRequest.method

Property Description	The server request http method. This property is read-only.
Type	http.Method enum
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Method',
  details: http.request.method
});
...
//Add additional code
```

ServerRequest.parameters

Property Description	The server request parameters. This property is read-only.
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members

Since	2015.2
-------	--------

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
log.debug({
    title: 'Server Request Parameters',
    details: http.request.parameters
});
...
//Add additional code
```

ServerRequest.url

Property Description	The server request URL. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerRequest
Sibling Object Members	ServerRequest Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
log.debug({
    title: 'Server Request URL',
    details: http.request.url
});
```

```
...
//Add additional code
```

http.ServerResponse

Object Description	Encapsulates the response to an incoming http request from an HTTP server. For example, a response from a Suitelet or RESTlet. For a complete list of this object's methods and properties, see ServerResponse Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.addHeader({
    name: 'Accept-Language',
    value: 'en-us',
});
...
//Add additional code
```

ServerResponse.addHeader(options)

Method Description	Method used to add a header to the response. If the same header has already been set, this method adds another line for that header. For example: <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <pre>Vary: 'Accept-Language' Vary: 'Accept-Encoding'</pre> </div>
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2
options.value	string	required	The value used to set the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
SSS_INVALID_HEADER	One or more headers are not valid.	The header name or value is invalid.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.addHeader({
  name: 'Accept-Language',
  value: 'en-us',
});
...
//Add additional code
```

ServerResponse.getHeader(options)

Method Description	Method used to return the value or values of a response header. If multiple values are assigned to the header name, the values are returned as an Array.
Returns	string string[]
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.getHeader({
    name: 'Accept-Language'
});
...
//Add additional code
```

ServerResponse.sendRedirect(options)

Method Description	Method used to set the redirect URL by resolving to a NetSuite resource. For example, you could use this method and your own parameters to make a redirect to a url for associated records, such as a redirect to a new sales order page for a particular entity.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The type of resource redirected to. Set this value using the http.RedirectType enum.	2015.2
options.identifier	number string	required	The primary ID for the resource. <ul style="list-style-type: none"> If redirecting to a media item (for example, an image or PDF file), pass in the file id. If redirecting to a record, pass in the record type using the record.Type enum. 	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> ■ If redirecting to a RESTlet, pass in the script ID as a number or string. ■ If redirecting to a tasklink, pass in the task ID. For a list of supported task IDs, see the help topic Task IDs. ■ If redirecting to a Suitelet, pass in the script ID. 	
options.id	number string	optional	The secondary ID for this resource. If the resource type is a Suitelet or RESTlet, pass in the deployment ID.	2015.2
options.editMode	boolean true false	optional	Applicable when redirecting to a record resource. Specifies whether to return a URL for a record in edit mode or view mode. The default value is <code>false</code> – returns the record in view mode and not edit mode.	2015.2
options.parameters	object	optional	Additional URL parameters as name/value pairs.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is missing. Note that this error is thrown if the script includes a typo in certain enums. For example, you see this error if you use <code>http.RedirectType.TASKLINK</code> instead of <code>http.RedirectType.TASK_LINK</code> in the <code>options.type</code> field.
SSS_INVALID_URL_CATEGORY	The options.type: {type} is not valid. Please use the <code>RedirectType</code> enum for supported types.	The script uses an unrecognizable string value for the <code>options.type</code> parameter. To avoid this error, use the http.RedirectType enum.
SSS_INVALID_TASK_ID	The task ID: {id} is not valid. Please refer to the documentation for a list of supported task IDs.	The type is set to <code>tasklink</code> , and an invalid task ID is input for <code>options.identifier</code> .
SSS_INVALID_RECORD_TYPE	Type argument {type} is not a valid record or is not available in your account. Please see the documentation for a list of supported record types.	The redirect type is set to <code>record</code> , and an invalid record type is input for <code>options.identifier</code> .
SSS_INVALID_SCRIPT_ID_1	You have provided an invalid script id or internal id: {id}	The type is set to <code>Suitelet</code> or <code>RESTlet</code> , and an invalid script ID or invalid deployment ID is input for <code>options.identifier</code> or <code>options.id</code> .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
```

```

...
myServerResponseObj.sendRedirect({
    type: http.RedirectType.RECORD,
    identifier: record.Type.SALES_ORDER,
    parameters: {entity: 8}
});
...
//Add additional code

```

ServerResponse.setHeader(options)

Method Description	Method used to set the value of a response header.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2
options.value	string	required	The value used to set the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
SSS_INVALID_HEADER	One or more headers are not valid.	The header name or value is invalid.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```

//Add additional code
...
serverResponse.setHeader({
    name: 'Accept-Language',
    value: 'en-us',
});
...

```

```
//Add additional code
```

ServerResponse.renderPdf(options)

Method Description	Method used to generate and render a PDF directly to the response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.xmlString	string	required	Content of the pdf.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.renderPdf({
    xmlString:'<?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>'
});
...
//Add additional code
```

ServerResponse.setCdnCacheable(options)

Method Description	Method used to set CDN caching for a period of time.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.type	enum	required	The value of the caching duration. Set using the http.CacheDuration enum.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.setCdnCacheable({
    type: http.CacheDuration.MAX
});
...
//Add additional code
```

ServerResponse.write(options)

Method Description	Method used to write information to the response.
	<div> Note: This method accepts only strings. To pass in a file, you can use ServerResponse.writeFile(options). </div>
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members

Since	2015.2
-------	--------

Parameters

Parameter	Type	Required / Optional	Description	Since
options.output	string	required	The output string being written.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.output is not a string.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.write({
    output: 'Hello World'
});
...
//Add additional code
```

ServerResponse.writeFile(options)

Method Description	Method used to write a file to the response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.file	file.File	required	A file.File Object that encapsulates the file to be written.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.isInline	boolean <code>true</code> <code>false</code>	optional	Determines whether the file is inline. If true, the file is inline. The default value is <code>false</code> .	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.file is not a file.File Object.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.writeFile({
    file: myFileObj,
    isInline: true
});
...
//Add additional code
```

ServerResponse.writeLine(options)

Method Description	Method used to write line information to the response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.output	string	required	The output string being written.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.output is not a string.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
serverResponse.writeLine({
    output: 'this is a sample string'
});
...
//Add additional code
```

ServerResponse.writePage(options)

Method Description	Method used to generate a page.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.pageObject	serverWidget.Assistant serverWidget.Form serverWidget.List	required	A standalone page object in the form of an assistant, form or list.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var myPageObj = serverWidget.createList({
    title: 'Simple List'
});

ServerResponse.writePage({
    pageObject: myPageObj
});
...
//Add additional code
```

ServerResponse.headers

Property Description	The server response headers. This property is read-only.
Type	Object If multiple values are assigned to one header name, the values are returned as an array. For example: <pre>{Vary: ['Accept-Language', 'Accept-Encoding']}</pre>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/http Module
Parent Object	http.ServerResponse
Sibling Object Members	ServerResponse Object Members
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
```

```

if serverResponse.headers.Content-Type === 'text/plain'
    return true

log.debug({
    title: "Server Response Headers",
    details: ServerResponse.headers
});
...
//Add additional code

```

http.get(options)

Method Description	Method used to send an HTTP GET request and return the response
Returns	http.ClientResponse
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTP URL being requested	2015.2
options.headers	object	optional	The HTTP headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```

//Add additional code
...
var response = http.get({
    url: 'http://www.google.com'
});
...

```

```
//Add additional code
```

http.get.promise(options)

Method Description	Method used to send an HTTP GET request asynchronously and return the response <div>  Note: For information about the parameters and errors thrown for this method, see http.get(options). For additional information on promises, see Promise Object. </div>
Returns	A http.ClientResponse object.
Synchronous Version	http.get(options)
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/http Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
http.get.promise({
  url: 'http://www.google.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Get Request: ',
    details: reason
  });
})
...
//Add additional code
```

http.delete(options)

Method Description	Method used to send an HTTP DELETE request and return the response.
---------------------------	---

	<div>  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs. </div> <div>  Note: This method does not include an <code>options.body</code> parameter. Postdata is not required when the HTTP method is a DELETE request. </div>
Returns	<code>http.ClientResponse</code>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.url</code>	string	required	The HTTP URL being requested	2015.2
<code>options.headers</code>	object	optional	The HTTP headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = http.delete({
    url: 'http://www.mytestwebsite.com'
});
...
//Add additional code
```

http.delete.promise(options)

Method Description	Method used to send an HTTP DELETE request asynchronously and return the response.
---------------------------	--

	 Note: For information about the parameters and errors thrown for this method, see http.delete(options) . For additional information on promises, see Promise Object .
Returns	A http.ClientResponse object
Synchronous Version	http.delete(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/http Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
http.delete.promise({
  url: 'http://www.mytestwebsite.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

http.request(options)

Method Description	Method used to send an HTTP request and return the response. <div>  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs. </div>
Returns	A http.ClientResponse object. For a complete list of this object's properties, see ClientResponse Object Members .

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.method	enum	required	The HTTP request method. Set using the http.Method enum.	2015.2
options.url	string	required	The HTTP URL being requested	2015.2
options.body	string object	optional	The POST data if the method is <code>POST</code> . <div> Note: If the method is <code>DELETE</code>, this body data is ignored.</div>	
options.headers	object	optional	An object containing request headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = http.request({
  method: http.Method.GET,
  url: 'http://www.google.com'
});
...
//Add additional code
```

http.request.promise(options)

Method Description	Method used to send an HTTP request asynchronously and return the response.
---------------------------	---

	 Note: For information about the parameters and errors thrown for this method, see http.request(options) . For additional information on promises, see Promise Object .
Returns	A http.ClientResponse object
Synchronous Version	http.request(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/http Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
http.request.promise({
  method: http.Method.GET,
  url: 'http://www.google.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

http.post(options)

Method Description	Method used to send an HTTP POST request and return the response. <div>  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs. </div>
Returns	A http.ClientResponse object

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTP URL being requested	2015.2
options.body	string object	required	The POST data.	2015.2
options.headers	object	optional	The HTTP headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = https.post({
  url: 'http://www.google.com',
  body: myPostDataObj
});
...
//Add additional code
```

http.post.promise(options)

Method Description	Method used to send an HTTP POST request asynchronously and return the response.  Note: For information about the parameters and errors thrown for this method, see http.post(options) . For additional information on promises, see Promise Object .
Returns	A http.ClientResponse object

Synchronous Version	http.post(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/http Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
http.post.promise({
  url: 'http://www.google.com',
  body: myPostDataObj
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

http.put(options)

Method Description	Method used to send an HTTP PUT request and return the response. <div> Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs.</div>
Returns	http.ClientResponse object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/http Module

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTP URL being requested	2015.2
options.body	string object	required	The PUT data.	2015.2
options.headers	object	optional	The HTTP headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = http.put({
  url: 'http://www.google.com',
  body: myDataObj,
  headers: headerObj
});
...
//Add additional code
```

http.put.promise(options)

Method Description	Method used to send an HTTP PUT request asynchronously and return the response.  Note: For information about the parameters and errors thrown for this method, see http.put(options) . For additional information on promises, see Promise Object .
Returns	http.ClientResponse object
Synchronous Version	http.put(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units

Module	N/http Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
http.put.promise({
  url: 'http://www.google.com',
  body: myDataObj,
  headers: headerObj
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

http.CacheDuration

Enum Description	<p>Holds the string values for supported cache durations. This enum is used to set the value of the <code>ServerResponse.setCdnCacheable(options)</code> property.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/http Module

Values

- LONG
- MEDIUM
- SHORT

UNIQUE

Syntax

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
ServerResponse.setCdnCacheable({
    type: http.CacheDuration.MAX
});
...
//Add additional code
```

http.Method

Enum Description	<p>Holds the string values for supported HTTP requests. This enum is used to set the value of <code>http.request(options)</code> and <code>ServerRequest.method</code>.</p> <div> <p>Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</p> </div>
Supported Script Types	<p>Server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/http Module

Values

- DELETE
- GET
- HEAD
- PUT
- POST

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module](#).

```
//Add additional code
...
var response = http.request({
```

```

method: http.Method.GET,
url: 'http://www.google.com'
});
...
//Add additional code

```

http.RedirectType

Enum Description	<p>Holds the string values for supported NetSuite resources that you can redirect to. This enum is used to set the value of the <code>type</code> argument for <code>ServerResponse.sendRedirect(options)</code>.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/http Module

Values

- MEDIA_ITEM
- RECORD
- RESTLET
- SUITELET
- TASK_LINK

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```

//Add additional code
...
myServerResponseObj.sendRedirect({
  type: http.RedirectType.RECORD,
  identifier: record.Type.SALES_ORDER,
  parameters: {entity: 6}
});...
//Add additional code

```

N/https Module

Load the https module when you need to manage content sent to a third party via HTTPS calls. This module encapsulates all the functionality of the [N/http Module](#), but does not allow the HTTP protocol. You can make HTTPS calls from client and server-side scripts.

SecureString functionality is supported only in server-side scripts. You can also use this functionality to perform various string transformations using methods that hash, encode, or append another string.

You can use this module to encode binary content or access a handle to the value in a NetSuite credential field.

When the https module is used, SuiteScript also loads the [N/crypto Module](#) and [N/encode Module](#).



Important: Use TLS 1.2 for https requests. For more information, see the help topics [FAQ: Transport Layer Security \(TLS\) Deprecations](#), specifically [SuiteScript and TLS](#).



Important: NetSuite supports the same list of trusted third-party certificate authorities (CAs) as Microsoft. For a list of these CAs, see <http://social.technet.microsoft.com/wiki/contents/articles/31634.microsoft-trusted-root-certificate-program-participants-v-2016-april.aspx>

- [N/https Module Members](#)
- [SecureString Object Members](#)
- [ClientResponse Object Members](#)
- [ServerResponse Object Members](#)
- [ServerRequest Object Members](#)
- [N/https Module Script Sample](#)

General HTTPS Header Blacklist

Be aware that certain headers cannot be set manually when using https module methods. If a script attempts to set values for any of the following headers, the values are discarded. These headers are described in the following table.

<ul style="list-style-type: none"> ■ Connection ■ Content-Length ■ Host ■ JSESSIONID ■ Trailer 	<ul style="list-style-type: none"> ■ Transfer-Encoding ■ Upgrade ■ Via
---	---

Suitelet Response HTTPS Header Blacklist

In addition to the headers described in [General HTTPS Header Blacklist](#), certain headers cannot be set manually when interacting with the [https.ServerResponse](#) objects sent by Suitelets. If a script attempts to set values for any of these headers, the system throws an SSS_INVALID_HEADER error. These headers are described in the following table.

<ul style="list-style-type: none"> ■ Access-Control-Allow-Origin ■ Allow ■ Connection ■ Content-Length ■ Content-Location ■ Content-MD5 ■ Content-Range 	<ul style="list-style-type: none"> ■ Date ■ JSESSIONID ■ Location ■ Proxy-Authenticate ■ Retry-After ■ Server ■ Trailer 	<ul style="list-style-type: none"> ■ Warning ■ WWW-Authenticate
--	--	---

	■ Via	
--	-------	--

N/https Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	https.SecureString	Object	Server-side scripts	Encapsulates data that may be sent to a third-party via an HTTPS call.
	https.ClientResponse	read-only Object	Server-side scripts	Encapsulates the response to an HTTPS client request.
	https.ServerRequest	read-only Object	Server-side scripts	Encapsulates the HTTPS request information sent to an HTTPS server. For example, a request received by a Suitelet or RESTlet.
	https.ServerResponse	Object	Server-side scripts	Encapsulates the response from an HTTPS server to an HTTPS request. For example, a response from a Suitelet or RESTlet.
Method	https.createSecureKey(options)	Object	Server-side scripts	Creates a key for the contents of a credential field.
	https.createSecureKey.promise(options)	Object	Client scripts	Creates a key asynchronously for the contents of a credential field.
	https.createSecureString(options)	Object	Server-side scripts	Creates an https.SecureString object.
	https.createSecureString.promise(options)	Object	Client scripts	Creates an https.SecureString object asynchronously.
	https.delete(options)	https.ClientResponse	Server-side scripts	Sends an HTTPS DELETE request and returns the response.
	https.delete.promise(options)	https.ClientResponse	Client scripts	Sends an HTTPS DELETE request asynchronously and returns the response.
	https.get(options)	https.ClientResponse	Server-side scripts	Sends an HTTPS GET request and returns the response.
	https.get.promise(options)	https.ClientResponse	Client scripts	Sends an HTTPS GET request asynchronously and returns the response.
	https.post(options)	https.ClientResponse	Server-side scripts	Sends an HTTPS POST request and returns the response.
	https.post.promise(options)	https.ClientResponse	Client scripts	Sends an HTTPS POST request asynchronously and returns the response.
	https.put(options)	https.ClientResponse	Server-side scripts	Sends an HTTPS PUT request and returns the response.
	https.put.promise(options)	https.ClientResponse	Client scripts	Sends an HTTPS PUT request asynchronously and returns the response.
	https.request(options)	https.ClientResponse	Server-side scripts	Sends an HTTPS request and returns the response.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				If a request fails, an error. <code>SuiteScriptError</code> is thrown.
	<code>https.request.promise(options)</code>	<code>https.ClientResponse</code>	Client scripts	Sends an HTTPS request asynchronously and returns the response. If a request fails, a <code>Promise.reject</code> is thrown with a <code>parameter Error</code> .
Enum	<code>https.CacheDuration</code>	enum	Server-side scripts	Holds the string values for supported cache durations. This enum is used to set the value of the <code>ServerResponse.setCdnCacheable(options)</code> property.
	<code>https.Encoding</code>	enum	Server-side scripts	Holds the string values for supported encoding types. This enum is used to set the value of parameters in <code>SecureString.appendString(options)</code> , <code>SecureString.convertEncoding(options)</code> , <code>https.createSecureString(options)</code> .
	<code>https.Method</code>	enum	Server-side scripts	Holds the string values for supported HTTP requests. This enum is used to set the value of <code>https.request(options)</code> and <code>ServerRequest.method</code> .
	<code>https.RedirectType</code>	enum	Server-side scripts	Holds the string values for supported NetSuite resources to which you can redirect. This enum is used to set the value of the <code>type</code> argument for <code>ServerResponse.sendRedirect(options)</code> .

SecureString Object Members

The following members are called on `https.SecureString`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>SecureString.appendSecureString(options)</code>	<code>https.SecureString</code>	Server-side scripts	Appends a passed in <code>https.SecureString</code> to another <code>https.SecureString</code> .
	<code>SecureString.appendString(options)</code>	<code>https.SecureString</code>	Server-side scripts	Appends a passed in string to a <code>https.SecureString</code> .
	<code>SecureString.convertEncoding(options)</code>	<code>https.SecureString</code>	Server-side scripts	Changes the encoding of a <code>https.SecureString</code> .
	<code>SecureString.hash(options)</code>	<code>https.SecureString</code>	Server-side scripts	Produces the <code>https.SecureString</code> as a hash.
	<code>SecureString.hmac(options)</code>	<code>https.SecureString</code>	Server-side scripts	Produces the <code>https.SecureString</code> as an hmac.

ClientResponse Object Members

The following members are called on [http.ClientResponse](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	ClientResponse.body	read-only string	Server-side scripts	The response body.
	ClientResponse.code	read-only number	Server-side scripts	The response code.
	ClientResponse.headers	read-only Object	Server-side scripts	The response body.

ServerRequest Object Members

The following members are called on the [http.ServerRequest](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	ServerRequest.getLineCount(options)	number	Server-side scripts	Returns the number of lines in a sublist.
	ServerRequest.getSublistValue(options)	string	Server-side scripts	Returns the value of a sublist line item.
Property	ServerRequest.body	read-only string	Server-side scripts	The server request body
	ServerRequest.files	read-only Object	Server-side scripts	The server request files.
	ServerRequest.headers	read-only Object	Server-side scripts	The server request headers.
	ServerRequest.method	https.Method enum	Server-side scripts	The HTTPS method for the server request.
	ServerRequest.parameters	read-only Object	Server-side scripts	The server request parameters.
	ServerRequest.url	read-only string	Server-side scripts	The server request URL.

ServerResponse Object Members

The following members are called on the [http.ServerResponse](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	ServerResponse.addHeader(options)	void	Server-side scripts	Adds a header to the response
	ServerResponse.getHeader(options)	void	Server-side scripts	Returns the value of a response header
	ServerResponse.renderPdf(options)	void	Server-side scripts	Generates and renders a PDF directly to the response

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	ServerResponse.sendRedirect(options)	void	Server-side scripts	Sets the redirect URL by resolving to a NetSuite resource
	ServerResponse.setCdnCacheable(options)	void	Server-side scripts	Sets CDN caching for a period of time.
	ServerResponse.setHeader(options)	void	Server-side scripts	Sets the value of a response header.
	ServerResponse.write(options)	void	Server-side scripts	Writes information (text/xml/html) to the response.
	ServerResponse.writeFile(options)	void	Server-side scripts	Writes a file to the response.
	ServerResponse.writeLine(options)	void	Server-side scripts	Writes line information (text/xml/html) to the response.
	ServerResponse.writePage(options)	void	Server-side scripts	Generates a page.
Property	ServerResponse.headers	Object	Server-side scripts	The server response headers.

N/https Module Script Sample

The following example uses a GUID to generate a secure token and a secret key. Note this example is meant to show how to use the APIs but will not actually work in the debugger because the GUID does not exist in your account. Please try the Suitelet example for a more complete usage. To run, this sample in the debugger, you must replace the GUID with one specific to your account.



Note: This sample script uses a `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

```
/**
 * @NApiVersion 2.x
 */
require(['N/runtime', 'N/https', 'N/crypto'],
function(http, https, crypto) {
    function createSecureString() {
        var passwordGuid = '{284CFB2D225B1D76FB94D150207E49DF}';
        var secureToken = https.createSecureString({
            input: passwordGuid
        });
        var secretKey = https.createSecretKey({
            input: passwordGuid
        });
        secureToken = secureToken.hmac({
            algorithm: crypto.HashAlg.SHA256,
            key: secretKey
        });
    }
    createSecureString();
});
```

```
});
```

The following example is a Suitelet sample that shows creating a form field that generates a GUID.

For more information about credential fields, see [Form.addCredentialField\(options\)](#).



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.

This sample uses the `define` function. The NetSuite Debugger cannot step through a `define` function. If you need to step through your code in the NetSuite Debugger, you must use a `require` function.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget', 'N/https'],
function(ui, https) {
    function onRequest(option) {
        if (option.request.method === 'GET') {
            var form = ui.createForm({
                title: 'Password Form'
            });
            var credField = form.addCredentialField({
                id: 'password',
                label: 'Password',
                restrictToDomains: ['system.netsuite.com'],
                restrictToCurrentUser: false,
                restrictToScriptIds: 'customscript_my_script'
            });
            credField.maxLength = 64;
            form.addSubmitButton();
            option.response.writePage({
                pageObject: form
            });
        } else {
            // Request to an existing suitelet with credentials
            var passwordGuid = option.request.parameters.password;
            var baseUrl = 'https://system.netsuite.com/app/site/hosting/scriptlet.nl?script=995&deploy=1&compid=1326288&h=397b6b0d3a4170b26735';
            var url = baseUrl + '&pwd={' + passwordGuid + '}';
            var secureStringUrl = https.createSecureString({
                input: url
            });
            var secureStringPWD = https.createSecureString({
                input: '{ ' + passwordGuid + '}'
            });
            var headers = {
                'pwd': secureStringPWD
            };
            var response = https.get({
                credentials: [passwordGuid],
```

```

        url: secureStringUrl,
        headers: headers
    });
}
}
return {
    onRequest: onRequest
};
});

```

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

https.SecureString

Object Description	Encapsulates a request string, such as a fragment of sensitive data that is going to be sent to a third party. This object is needed when you create a securestring, put your data in it, and encode it a particular way. For a complete list of this object's methods, see SecureString Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```

//Add additional code
...
function createSecureString() {
    var passwordGuid = '{284CFB2D225B1D76FB94D150207E49DF}';
    var secureToken = https.createSecureString({
        input: passwordGuid
    });
    ...
//Add additional code

```

SecureString.appendSecureString(options)

Method Description	Method used to append a passed in https.SecureString to another https.SecureString .
Returns	https.SecureString
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.secureString	https.SecureString object	required	The https.SecureString to append.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
string1.appendSecureString({
    secureString: secureString2
});
...
//Add additional code
```

SecureString.appendString(options)

Method Description	Method used to append a passed string to an https.SecureString .
Returns	https.SecureString
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.input	string	required	The string to append.

Parameter	Type	Required / Optional	Description
<code>options.inputEncoding</code>	https.Encoding enum	required	The encoding of the string that is being appended.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
string1.appendString({
  input: '48656c6c6f20776f726c640d0a',
  encoding: https.Encoding.HEX});
...
//Add additional code
```

SecureString.convertEncoding(options)

Method Description	Changes the encoding of a https.SecureString
Returns	https.SecureString
Governance	None
Module	N/https Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.toEncoding</code>	https.Encoding	required	The encoding to apply to the returned string.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code ...
https.convertEncoding({
  toEncoding: https.Encoding.HEX
});
...
```

```
//Add additional code
```

SecureString.hash(options)

Method Description	Hashes an https.SecureString object
Returns	https.SecureString
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.algorithm</code>	crypto.Hash enum	required	The hash algorithm. Set the value using the crypto.Hash enum.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
secureString = secureString.hash({
    algorithm: crypto.HashAlg.SHA256
});
...
//Add additional code
```

SecureString.hmac(options)

Method Description	Produces the <code>securestring</code> as an hmac.
Returns	https.SecureString
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.algorithm	crypto.Hash enum	required	The hash algorithm. Set by the crypto.Hash enum.
options.key	crypto.SecretKey	required	A key returned from https.createSecureKey(options) .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
secureToken = secureToken.hmac({
  algorithm: crypto.HashAlg.SHA256,
  key: secretKey
});
...
//Add additional code
```

https.createSecureKey(options)

Method Description	Creates and returns a crypto.SecretKey object. This method can take a GUID. Use Form.addCredentialField(options) to generate a value. You can put the key in your secure string. SuiteScript decrypts the value (key) and sends it to the server
Returns	crypto.SecretKey
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.encoding	https.Encoding enum	optional	Specifies the encoding for the SecureKey.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.guid	string	required	A GUID used to generate a secret key. The GUID can resolve to either data or metadata.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var secretKey = https.createSecretKey({
    encoding: https.Encoding.HEX,
    guid: '284CFB2D225B1D76FB94D150207E49DF'
});
...
//Add additional code
```

https.createSecureKey.promise(options)

Method Description	Creates and returns a crypto.SecretKey object asynchronously.
	Note: For information about the parameters and errors thrown for this method, see https.createSecureKey(options) . For additional information on promises, see Promise Object .
Returns	A crypto.SecretKey object
Synchronous Version	https.createSecureKey(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
var secretKey = https.createSecretKey.promise({
    encoding: https.Encoding.HEX,
    guid: '284CFB2D225B1D76FB94D150207E49DF'
});
...
//Add additional code
```

```
});

...
//Add additional code
```

https.createSecureString(options)

Method Description	Creates and returns an https.SecureString .
Returns	https.SecureString
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.input	string	required	The string to convert to a securestring.	Release 15 Version 2
options.inputEncoding	https.Encoding enum	optional	Identifies the encoding that the input string uses. The default value is <code>UTF_8</code>	Release 15 Version 2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var secureToken = https.createSecureString({
    input: passwordGuid
});
...
//Add additional code
```

https.createSecureString.promise(options)

Method Description	Creates and returns an https.SecureString asynchronously.
---------------------------	---

	 Note: For information about the parameters and errors thrown for this method, see https.createSecureString(options) . For additional information on promises, see Promise Object .
Returns	SecureTokenResolver
Synchronous Version	https.createSecureString(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
var secureToken = https.createSecureString.promise({
  input: passwordGuid
});
...
//Add additional code
```

https.ClientResponse

Object Description	Encapsulates the response to an HTTPS client request. This object is read-only. For a complete list of this object's properties, see ClientResponse Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var clientResponse = https.get({
  url: 'https://www.testwebsite.com'
});
```

```
...
//Add additional code
```

ClientResponse.body

Property Description	The client response body. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = https.get({
    url: 'https://www.testwebsite.com'
});
log.debug({
    title: 'Client Response Body',
    details: https.response.body
});
...
//Add additional code
```

ClientResponse.code

Property Description	The client response code. This property is read-only.
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = https.get({
    url: 'https://www.testwebsite.com'
});
log.debug({
    title: 'Client Response Code',
    details: https.response.code
});
...
//Add additional code
```

ClientResponse.headers

Property Description	The response header or headers. This property is read-only.
Type	object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/http Module Script Sample](#).

```
//Add additional code
...
var response = https.get({
    url: 'https://www.testwebsite.com'
});
```



```
log.debug({
  title: 'Client Response Header',
  details: https.response.headers
});
...
//Add additional code
```

https.ServerRequest

Object Description	Encapsulates the incoming https request information for an HTTPS server. This object is read-only. For a complete list of this object's methods and properties, see ServerRequest Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [https.ServerRequest](#).

```
//Add additional code
...
serverRequest.getLineCount({
  group: 'sublistId'
});
...
//Add additional code
```

ServerRequest.getLineCount(options)

Method Description	Method used to return the number of lines in a sublist.
Returns	The number of lines in a sublist as a number.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.group	string	required	The sublist internal ID.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverRequest.getLineCount({
    group: 'sublistId'
});
...
//Add additional code
```

ServerRequest.getSublistValue(options)

Method Description	Method used to return the value of a sublist line item.
Returns	The value of the sublist line item as a string.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.group	string	required	The sublist internal ID.	2015.2
options.name	string	required	The name of the field.	2015.2
options.line	string	required	The sublist line number.	2015.2



Note: Sublist index starts at 0.

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverRequest.getSublistValue({
    group: 'item',
    name: 'amount',
    line: '2'
});
...
//Add additional code
```

ServerRequest.body

Property Description	The server request body. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
log.debug({
    title: 'Server Request Body',
    details: https.request.body
});
...
//Add additional code
```

ServerRequest.files

Property Description	The server request files.
----------------------	---------------------------

	This property is read-only.
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippets show the syntax for this member. They are not functional examples. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Files',
  details: https.request.files
});
...
//Add additional code
```

```
var file = request.files['file_id'];
```

ServerRequest.headers

Property Description	<p>This object represents a series of key/value pairs. Each pair represents a server request header name and its value.</p> <p>Typically, this object encapsulates two iterations of each header name: one in lower case and another in title case. This behavior is designed so that you can use either lower case or title case when you reference a header. However, the existence of title-case iterations of header names is not guaranteed. For best results, refer to header names using all lower-case letters (and hyphens, when applicable).</p> <p>This property is read-only.</p> <div> <p>Important: The server request headers and their values are subject to change. If you use these headers in your scripts, you are responsible for testing them to make sure that they contain the information you need. For example, when making an HTTP call to a Suitelet, some headers might be filtered out. Filtering can occur if the headers affect how NetSuite processes the request internally. These filtered headers are not available to the Suitelet, so you should test to see whether a header was filtered out. If so, use a different header instead.</p> </div>
Type	Object

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Server Request Headers',
  details: https.request.headers
});
...
//Add additional code
```

ServerRequest.method

Property Description	The server request https method. This property is read-only.
Type	enum
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [Parameters](#).

```
//Add additional code
```

```

...
log.debug({
    title: 'Server Request Method',
    details: https.request.method
});
...
//Add additional code

```

ServerRequest.parameters

Property Description	The server request parameters. This property is read-only.
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```

//Add additional code
...
log.debug({
    title: 'Server Request Parameters',
    details: https.request.parameters
});
...
//Add additional code

```

ServerRequest.url

Property Description	The server request URL. This property is read-only.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
log.debug({
    title: 'Server Request URL',
    details: https.request.url
});
...
//Add additional code
```

https.ServerResponse

Object Description	Encapsulates the response to an incoming http request from an HTTP server. For example, a response from a Suitelet or RESTlet. For a complete list of this object's methods and properties, see ServerResponse Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.addHeader({
    name: 'Accept-Language',
    value: 'en-us',
});
...
```

```
//Add additional code
```

ServerResponse.addHeader(options)

Method Description	Method used to add a header to the response. If the same header has already been set, this method adds another line for that header. For example: <pre>Vary: 'Accept-Language' Vary: 'Accept-Encoding'</pre>
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2
options.value	string	required	The value used to set the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
SSS_INVALID_HEADER	One or more headers are not valid.	The header name or value is invalid.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.addHeader({
  name: 'Accept-Language',
  value: 'en-us',
});
...
//Add additional code
```


ServerResponse.getHeader(options)

Method Description	Method used to return the value or values of a response header. If multiple values are assigned to the header name, the values are returned as an Array.
Returns	string string[]
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.getHeader({
  name: 'Accept-Language'
});
...
//Add additional code
```

ServerResponse.sendRedirect(options)

Method Description	Method used to create a redirect URL that resolves to a NetSuite resource. For example, you could use this method to redirect to a new sales order page for a particular entity.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

 **Important:** All parameters must be prefixed with `custparam`.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The type of resource to which the script redirects. Use the https.RedirectType enum to set a value for this parameter.	2015.2
options.identifier	number string	required	The primary ID for this resource. The value you use varies depending on the value of options.type, as follows: <ul style="list-style-type: none"> MEDIA_ITEM — Use the internal ID of a file stored in the NetSuite File Cabinet. RECORD — Use the record.Type enum to identify the appropriate record type. RESTLET — Use the script ID from the script record of the appropriate RESTlet. SUITELET — Use the script ID from the script record of the appropriate Suitelet. TASK_LINK — Use the appropriate Task ID. Supported IDs are listed in Task IDs. 	2015.2
options.id	string	optional	The secondary ID for this resource. If the options.type parameter is set to SUITELET or RESTLET, use the deployment ID. If the options.type parameter is set to RECORD, you can use the internal ID of a specific record instance.	2015.2
options.editMode	boolean true false	optional	Applicable when redirecting to a record. Use the following values: <ul style="list-style-type: none"> true — returns the record in edit mode. false — returns the record in view mode. The default value is <code>false</code> .	2015.2
options.parameters	object	optional	Additional URL parameters as key-value pairs.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is missing. Note that this error is thrown if the script includes a typo in certain enums. For example, you see this error if you use <code>https.RedirectType.TASKLINK</code> instead of <code>https.RedirectType.TASK_LINK</code> in the options.type field.
SSS_INVALID_URL_CATEGORY	The options.type: {type} is not valid. Please use the	The script uses an unrecognizable string value for the options.type parameter. To avoid this error, use the https.RedirectType enum.

Error Code	Message	Thrown If
	RedirectType enum for supported types.	
INVALID_TASK_ID	The task ID: {id} is not valid. Please refer to the documentation for a list of supported task IDs.	The options.type parameter is set to TASK_LINK, and the script uses an invalid task ID for options.identifier. For a list of valid IDs, see the help topic Task IDs .
INVALID_RCRD_TYPE	The record type {type} is invalid.	The options.type parameter is set to RECORD, and the script uses an unrecognizable string value for options.identifier. To avoid this error, use the record.Type enum to identify the appropriate record type.
INVALID_ID	You have provided an invalid script id or internal id: {id}	The options.type parameter is set to RESTLET or SUITELET, and the script uses an invalid ID for options.identifier or options.id.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
myServerResponseObj.sendRedirect({
    type: https.RedirectType.RECORD,
    identifier: record.Type.SALES_ORDER,
    parameters: {entity: 8}
});
...
//Add additional code
```

ServerResponse.setHeader(options)

Method Description	Method used to set the value of a response header.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.name	string	required	The name of the header.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.value	string	required	The value used to set the header.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
SSS_INVALID_HEADER	One or more headers are not valid.	The header name or value is invalid.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.setHeader({
    name: 'Accept-Language',
    value: 'en-us',
});
...
//Add additional code
```

ServerResponse.renderPdf(options)

Method Description	Method used to generates and renders a PDF directly to the response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.xmlString	string	required	Content of the pdf.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.renderPDF({
    xmlString:'<?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>'
});
...
//Add additional code
```

ServerResponse.setCdnCacheable(options)

Method Description	Method used to set CDN caching for a period of time.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The value of the caching duration. Set using the https.CacheDuration .	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.setCdnCacheable({
    type: https.CacheDuration.MAX
});
```

```
...
//Add additional code
```

ServerResponse.write(options)

Method Description	Method used to write information to the response. Note: This method accepts only strings. To pass in a file, you can use <code>ServerResponse.writeFile(options)</code> .
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.output	string	required	The output string being written.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.output is not a string.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.write({
  output: 'Hello World'
});
...
//Add additional code
```

ServerResponse.writeFile(options)

Method Description	Method used to write a file to the response.
---------------------------	--

Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.file	file.File	required	A file.File Object that encapsulates the file to be written.	2015.2
options.isInline	boolean true false	optional	Determines whether the field is inline. If true, the file is inline.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.file is not a file.File Object.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.writeFile({
    file: myFileObj,
    isInline: true
});
...
//Add additional code
```

ServerResponse.writeLine(options)

Method Description	Method used to write line information to the response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module

Since	2015.2
-------	--------

Parameters

Parameter	Type	Required / Optional	Description	Since
options.output	string	required	The output string being written.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.
WRONG_PARAMETER_TYPE	{param name}	The value input for options.output is not a string.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
serverResponse.writeLine({
    output: 'this is a sample string'
});
...
//Add additional code
```

ServerResponse.writePage(options)

Method Description	Method used to generate a page.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/https Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description	Since
options.pageObject	serverWidget.Assistant serverWidget.Form serverWidget.List	required	A standalone page object in the form of an assistant, form or list.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var myPageObj = serverWidget.createList({
    title: 'Simple List'
});

ServerResponse.writePage({
    pageObject: myPageObj
});
...
//Add additional code
```

ServerResponse.headers

Property Description	The server response headers. This property is read-only.
Type	Object Note that If multiple values are assigned to one header name, the values are returned as an array. For example: <pre>{Vary: ['Accept-Language', 'Accept-Encoding']}</pre>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY_PROPERTY		You attempt to edit this property.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
```

```

...
if serverResponse.headers.Content-Type === 'text/plain'
return true

log.debug({
  title: "Server Response Headers",
  details: serverResponse.headers
});
...
//Add additional code

```

https.get(options)

Method Description	Method used to send an HTTPS GET request and return the response
Returns	https.ClientResponse
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTPS URL being requested	2015.2
options.headers	object	optional	The HTTPS headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```

//Add additional code
...
var response = https.get({
  url: 'https://www.testwebsite.com'
});
...

```

```
//Add additional code
```

https.get.promise(options)

Method Description	Method used to send an HTTPS GET request asynchronously and return the response <div>  Note: For information about the parameters and errors thrown for this method, see https.get(options). For additional information on promises, see Promise Object. </div>
Returns	A https.ClientResponse object.
Synchronous Version	https.get(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
https.get.promise({
  url: 'http://www.testwebsite.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Get Request: ',
    details: reason
  });
})
...
//Add additional code
```

https.delete(options)

Method Description	Method used to send an HTTPS DELETE request and returns the response.
---------------------------	---

	<div>  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs. </div> <div>  Note: This method does not include an <code>options.body</code> parameter. Postdata is not required when the HTTPS method is a DELETE request. </div>
Returns	https.ClientResponse
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTPS URL being requested	2015.2
options.headers	object	optional	The HTTPS headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var response = https.delete({
    url: 'http://www.mytestwebsite.com'
});
...
//Add additional code
```

https.delete.promise(options)

Method Description	Method used to send an HTTP DELETE request asynchronously and return the response.
---------------------------	--

	 Note: For information about the parameters and errors thrown for this method, see https.delete(options) . For additional information on promises, see Promise Object .
Returns	A https.ClientResponse object
Synchronous Version	https.delete(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/https Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
https.delete.promise({
  url: 'http://www.mytestwebsite.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

https.request(options)

Method Description	Method used to send an HTTPS request and return the response. <div>  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs. </div>
Returns	An https.ClientResponse Object. For a complete list of this object's properties, see ClientResponse Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	10 units
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.method	enum	required	The HTTPS request method. Set using the https.Method enum.	2015.2
options.url	string	required	The HTTPS URL being requested	2015.2
options.body	string object	optional	The POST data if the method is <code>POST</code> . <div> Note: If the method is <code>DELETE</code>, this body data is ignored.</div>	
options.headers	object	optional	An object containing request headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var response = https.request({
  method: https.Method.GET,
  url: 'https://www.testwebsite.com'
});
...
//Add additional code
```

https.request.promise(options)

Method Description	Method used to send an HTTP request asynchronously and return the response. <div> Note: For information about the parameters and errors thrown for this method, see https.request(options). For additional information on promises, see Promise Object.</div>
Returns	A https.ClientResponse object

Synchronous Version	https.request(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/https Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
https.request.promise({
  method: https.Method.GET,
  url: 'https://www.testwebsite.com'
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

https.post(options)

Method Description	Method used to send an HTTPS POST request and return the response.  Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs.
Returns	An https.ClientResponse Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/https Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTPS URL being requested	2015.2
options.body	string object	required	The POST data.	
options.headers	object	optional	The HTTPS headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var response = https.post({
  url: 'https://www.testwebsite.com',
  body: myPostDataObj
});
...
//Add additional code
```

https.post.promise(options)

Method Description	Method used to send an HTTPS POST request asynchronously and return the response.  Note: For information about the parameters and errors thrown for this method, see https.post(options) . For additional information on promises, see Promise Object .
Returns	A https.ClientResponse object
Synchronous Version	https.post(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
https.post.promise({
  url: 'https://www.testwebsite.com',
  body: myPostDataObj
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

https.put(options)

Method Description	Method used to send an HTTPS PUT request and return server response.
	Important: If negotiating a connection to the destination server exceeds 5 seconds, a connection timeout occurs. If transferring a payload to the server exceeds 45 seconds, a request timeout occurs.
Returns	An https.ClientResponse Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/https Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.url	string	required	The HTTPS URL being requested	2015.2
options.body	string object	required	The PUT data.	

Parameter	Type	Required / Optional	Description	Since
options.headers	object	optional	The HTTPS headers.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	Missing a required argument: {param name}	A required parameter is not passed.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var response = https.put({
  url: 'https://www.testwebsite.com',
  body: myDataObj,
  headers: headerObj
});
...
//Add additional code
```

https.put.promise(options)

Method Description	Method used to send an HTTPS PUT request asynchronously and return the response. <div>  Note: For information about the parameters and errors thrown for this method, see https.put(options). For additional information on promises, see Promise Object. </div>
Returns	https.ClientResponse object
Synchronous Version	https.put(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/https Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
```

```
...
https.put.promise({
  url: 'https://www.testwebsite.com',
  body: myDataObj,
  headers: headerObj
})
.then(function(response){
  log.debug({
    title: 'Response',
    details: response
  });
})
.catch(function onRejected(reason) {
  log.debug({
    title: 'Invalid Request: ',
    details: reason
  });
})
...
//Add additional code
```

https.CacheDuration

Enum Description	<p>Holds the string values for supported cache durations. This enum is used to set the value of the <code>ServerResponse.setCdnCacheable(options)</code> property.</p> <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module

Values

- LONG
- MEDIUM
- SHORT
- UNIQUE

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#)

```
//Add additional code
```

```

...
ServerResponse.setCdnCacheable({
    type: https.CacheDuration.MAX
});
...
//Add additional code

```

https.Encoding

Enum Description	Holds the string values for supported encoding values. <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/https Module

Values

- UTF_8
- BASE_16
- BASE_32
- BASE_64
- BASE_64_URL_SAFE
- HEX

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#)

https.Method

Enum Description	Holds the string values for supported HTTPS requests. This enum is used to set the value of https.request(options) and ServerRequest.method . <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/https Module
--------	----------------

Values

- DELETE
- GET
- HEAD
- PUT
- POST

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
var response = https.request({
  method: https.Method.GET,
  url: 'https://www.testwebsite.com'
});
...
//Add additional code
```

https.RedirectType

Enum Description	<p>Holds the string values for supported NetSuite resources to which you can redirect. This enum is used to set the value of the <code>type</code> argument for <code>ServerResponse.sendRedirect(options)</code>.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/https Module

Values

Value	Description
MEDIA_ITEM	A file in the NetSuite File Cabinet
RECORD	A NetSuite record.
RESTLET	A deployed RESTlet.
SUITELET	A deployed Suitelet.

Value	Description
TASK_LINK	A page in NetSuite, as defined by a valid Task ID.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/https Module Script Sample](#).

```
//Add additional code
...
myServerResponseObj.sendRedirect({
    type: https.RedirectType.RECORD,
    identifier: record.Type.SALES_ORDER,
    parameters: {entity: 6}
});...
//Add additional code
```

N/log Module

Use the log module to access methods for logging script execution details.

The log methods can be accessed globally or by loading this module. Load the N/log module when you want to manually access its members, such as for testing purposes. For more information about global objects, see [SuiteScript 2.0 Global Objects](#).

- [N/log Module Members](#)
- [N/log Module Guidelines](#)
- [Using Log Levels](#)
- [Viewing Script Execution Logs](#)
- [log Module Script Sample](#)

N/log Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	log.audit(options)	Void	Client and server-side scripts	Logs an entry of type AUDIT to the Execution Log tab of the script deployment for the current script.
	log.debug(options)	Void	Client and server-side scripts	Logs an entry of type DEBUG to the Execution Log tab of the script deployment for the current script.
	log.emergency(options)	Void	Client and server-side scripts	Logs an entry of type EMERGENCY to the Execution Log tab of the script deployment for the current script.
	log.error(options)	Void	Client and server-side scripts	Logs an entry of type ERROR to the Execution Log tab of the script deployment for the current script.

N/log Module Guidelines

- NetSuite governs the amount of logging that can be done in any specific 60 minute time period. A company is allowed to make up to 100,000 log object method calls across **all** of their scripts. Script owners are notified if NetSuite detects that one script is logging excessively and automatically adjusts the log level.
- NetSuite purges system errors older than **60 days** and user-generated logs older than **30 days**. Because log persistence is not guaranteed, NetSuite recommends using custom records if you want to store script execution logs for extended periods.
- The Execution Log tab also lists notes returned by NetSuite such as error messages. For more information, see [N/error Module](#).
- If you deploy a client script to a form using [Form.clientScriptFileId](#) or [Form.clientScriptModulePath](#), using the N/log module adds the logs to the deployment of the parent script. The parent script can be either a beforeLoad user event script or a [SuiteScript 2.0 Suitelet Script Type](#).
- When an object (that is not a string) is passed to a log object method, NetSuite runs [JSON.stringify\(obj\)](#) on any values that are passed as the `details` parameter and equal a JavaScript object.

```
...
// log.debug(rec) //Shows the JSON representation of the current values in a record object
var id = rec.save();
...
```

Using Log Levels

Use the log methods along with the **Log Level** field on the Script Deployment to determine whether to log an entry on the **Execution Log** subtab. If a log level is defined on a Script Deployment, then only log Object method calls with a log type equal to or greater than this log level will be logged. This is useful during the debugging of a script or for providing useful execution notes for auditing or tracking purposes.

Log levels and log Object methods act as a filter on the amount of information logged. The following log levels are supported:

Log Level	Description
Debug	Shows all Audit, Error, and Emergency information on the Execution Log tab. This type of logging is suitable only for testing scripts. To avoid excessive logging, the debug log level is not recommended for active scripts in production.
Audit	Shows a record of events that have occurred during the processing of the script (for example, "A request was made to an external site.").
Error	Shows only unexpected script errors.
Emergency	Shows only the most critical errors in the script log.

Viewing Script Execution Logs

To view logs for a specific script, see the Execution Log subtab of a Script Deployment record. These logs are not guaranteed to persist for 30 days.

To view script execution log details for various scripts, go to Customization > Scripting > Script Execution Logs Customization > Scripting > Script Execution Logs. This list of script execution logs is an enhanced repository that stores all log details for 30 days.

On this page, you can perform the following tasks:

- Search for specific logs using filter options, such as log level, execution date range, and script name.

Note: The log list shows 10,000 entries at a time for a given filtered criteria. Users can view other logs by using the Date and Script filter options.

- Download the list as a CSV file or an Excel spreadsheet.
- Print the list.

When you debug a script in the SuiteScript Debugger, log details appear on the Execution Log tab of the SuiteScript Debugger. These details do not appear on the Script Deployment page for the script.

log Module Script Sample

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.x
 */
require(['N/log'],
function(myLog) {
    var myObject = {
        name: 'Jane',
        id: '123'
    };
    myLog.debug({
        title: 'hello!'
    });
    myLog.debug({
        title: 'hello!',
        details: 'world'
    });
    myLog.debug({
        title: 'myObj',
        details: myObject
    });
});
```

log.audit(options)

Method Description	Logs an entry of type AUDIT to the Execution Log tab of the script deployment for the current script. This entry will not appear on the Execution Log tab if the Log Level field for the script deployment is set to ERROR or above. Use this method for scripts in production.
Returns	void

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	Amount of logging in any 60 minute period is limited. See N/log Module Guidelines .
Module	N/log Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
title	string	Optional	String to appear in the Title column on the Execution Log tab of the script deployment. Maximum length is 99 characters. If you set this value to null, an empty string ("), or omit it, the word "Untitled" appears for the log entry.	2016.1
details	any	Required	You can pass any value for this parameter. If the value is a JavaScript Object type, <code>JSON.stringify(obj)</code> is called on the object before displaying the value. NetSuite truncates any resulting string over 3999 characters.	2016.1

Syntax

The following code snippet shows the syntax for this method.

```
//Add additional code
...
var var1 = 'value';
log.audit({
  title: 'Audit Entry',
  details: 'Value of var1 is: ' + var1
});
...
//Add additional code
```

log.debug(options)

Method Description	Logs an entry of type DEBUG to the Execution Log tab of the script deployment for the current script. This entry does not appear on the Execution Log tab if the Log Level field for the script deployment is set to AUDIT or above. Use this method for scripts in development.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	Amount of logging in any 60 minute period is limited. See N/log Module Guidelines .
Module	N/log Module
Since	2016.1

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
title	string	Required	String to appear in the Title column on the Execution Log tab of the script deployment. Maximum length is 99 characters. If you set this value to null, an empty string (""), or omit it, the word "Untitled" appears for the log entry.	2016.1
details	any	Optional	You can pass any value for this parameter. If the value is a JavaScript object type, <code>JSON.stringify(obj)</code> is called on the object before displaying the value. NetSuite truncates any resulting string over 3999 characters.	2016.1

Syntax

The following code snippet shows the syntax for this method.

```
//Add additional code
...
var var1 = 'value';
log.debug({
  title: 'Debug Entry',
  details: 'Value of var1 is: ' + var1
});
...
//Add additional code
```

log.emergency(options)

Method Description	Logs an entry of type EMERGENCY to the Execution Log tab of the script deployment for the current script. Use this method for scripts in production.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	Amount of logging in any 60 minute period is limited. See N/log Module Guidelines .
Module	N/log Module
Since	2016.1

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
title	string	Optional	String to appear in the Title column on the Execution Log tab of the script deployment.	2016.1

Parameter	Type	Required / Optional	Description	Since
			Maximum length is 99 characters. If you set this value to null, an empty string (''), or omit it, the word "Untitled" appears for the log entry.	
details	any	Required	You can pass any value for this parameter. If the value is a JavaScript Object type, <code>JSON.stringify(obj)</code> is called on the object before displaying the value. NetSuite truncates any resulting string over 3999 characters.	2016.1

Syntax

The following code snippet shows the syntax for this method.

```
//Add additional code
...
var var1 = 'value';
log.emergency({
  title: 'Emergency Entry',
  details: 'Value of var1 is: ' + var1
});
...
//Add additional code
```

log.error(options)

Method Description	Logs an entry of type ERROR to the Execution Log tab of the script deployment for the current script. This entry will not appear on the Execution Log tab if the Log Level field for the script deployment is set to EMERGENCY or above. Use this method for scripts in production.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	Amount of logging in any 60 minute period is limited. See N/log Module Guidelines .
Module	N/log Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
title	string	Optional	String to appear in the Title column on the Execution Log tab of the script deployment. Maximum length is 99 characters. If you set this value to null, an empty string (''), or omit it, the word "Untitled" appears for the log entry.	2016.1

Parameter	Type	Required / Optional	Description	Since
details	any	Required	You can pass any value for this parameter. If the value is a JavaScript object type, <code>JSON.stringify(obj)</code> is called on the object before displaying the value. NetSuite truncates any resulting string over 3999 characters.	2016.1

Syntax

The following code snippet shows the syntax for this method.

```
//Add additional code
...
var var1 = 'value';
log.error({
    title: 'Error Entry',
    details: 'Value of var1 is: ' + var1
});
...
//Add additional code
```

N/plugin Module

Load the N/plugin module to load custom plug-in implementations. For additional information, see the help topic [Custom Plug-ins](#).



Important: You cannot use the SuiteScript Debugger to debug a script on demand that uses the N/plugin module. You must use deployed debugging. To use deployed debugging, you must complete the steps described in [Adding a Script that Instantiates a Custom Plug-in to NetSuite](#). For the complete process on creating a custom plugin, see the help topic [Custom Plug-in Development](#). For additional information about on demand and deployed debugging, see the help topic [Using the SuiteScript Debugger](#).

- [N/plugin Module Members](#)
- [N/plugin Module Script Samples](#)

N/plugin Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>plugin.findImplementations(options)</code>	string[]	Server-side scripts	Returns the script IDs of custom plug-in type implementations.
Method	<code>plugin.loadImplementation(options)</code>	Object	Server-side scripts	Instantiates an implementation of the custom plug-in type.

N/plugin Module Script Samples

To test the following samples, you need a custom plugin type with Script ID:

`customscript_magic_plugin` and an interface with a single method: `int doTheMagic(int, int)`.



Important: You cannot use the SuiteScript Debugger to debug a script on demand that uses the N/plugin module. You must use deployed debugging. To use deployed debugging, you must complete the steps described in [Adding a Script that Instantiates a Custom Plug-in to NetSuite](#). For the complete process on creating a custom plugin, see the help topic [Custom Plug-in Development](#). For additional information about on demand and deployed debugging, see the help topic [Using the SuiteScript Debugger](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following example shows an implementation of the interface:

```
/**
 * @NApiVersion 2.0
 * @NScriptType pluginTypeimpl
 */
define(function() {
  return {
    doTheMagic: function (operand1, operand2)
    {
      return operand1 + operand2;
    }
  }
});
```

The following Suitelet example iterates through all implementations of the custom plugin type `customscript_magic_plugin`.



Important: The Suitelet script record must specify the plugin type under Custom Plug-in Types in order for it to recognize the plug-in.

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

```

plugin.findImplementations(options)

Method Description	Returns the script IDs of custom plug-in type implementations. Returns an empty list when there is no custom plug-in type with the script ID available for the executing script.
Returns	A string[] containing a list of custom plug-in implementation script IDs.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	
Module	N/config Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The script ID of the custom plug-in type.	2016.1
options.includeDefault	boolean true false	optional	The default value is true, indicating that the default implementation should be included in the list.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/plugin Module Script Samples](#).

```

//Add additional code
...
var impls = plugin.findImplementations({
  type: 'customscript_sample_plugin'
});
...
//Add additional code

```

plugin.loadImplementation(options)

Method Description	Instantiates an implementation of the custom plugin type. Returns the implementation which is currently selected in the UI (Manage Plug-ins page) when no implementation ID is explicitly provided.
Returns	An Object implementing the custom plug-in type.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	
Module	N/config Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The script ID of the custom plug-in type.	2016.1
options.implementation	string	optional	The script ID of the custom plug-in implementation.	2016.1

Error Code	Thrown If
UNABLE_TO_FIND_IMPLEMENTATION_1_FOR_PLUGIN_2	Either there is no such implementation of the provided plug-in type, or the plug-in type does not exist.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/plugin Module Script Samples](#).

```
//Add additional code
...
var pl = plugin.loadImplementation({
  type: 'customscript_sample_plugin'
});
...
//Add additional code
```

N/portlet Module

Load the portlet module to resize or refresh a form portlet. See [SuiteScript 2.0 Portlet Script Type](#).

- [N/portlet Module Members](#)
- [N/portlet Module Script Sample](#)

N/portlet Module Members

Member Type	Name	Return Type	Supported Script Types	Description
Method	portlet.resize	void	Client scripts	Resizes a form portlet immediately.
	portlet.refresh	void	Client scripts	Refreshes a form portlet immediately.

N/portlet Module Script Sample

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 resize function to adjust the height and width of the portlet based on the values of the text fields.

The sample also shows how to create a button that uses the refresh function. When pressed, the portlet is updated to show the current date.

For more information about how a portlet is displayed on the NetSuite dashboard, see [SuiteScript 2.0 Portlet Script Type](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @NApiVersion 2.0
 * @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.defaultValu = resizeLink.defaultValu = "<a id='resize_link'
onclick=\"require(['SuiteScripts/portletApiTestHelper'], function(portletApiTestHelper)
{portletApiTestHelper.changeSizeAndResizePortlet(); }) \" href='#'>Resize</a><br>";
}

function setComponentsForRefresh() {
    var textField = portletObj.addField({
        id: 'refresh_output',
        type: 'text',
        label: 'Date.now().toString()'
    });
    textField.defaultValu = Date.now().toString();
    var refreshLink = portletObj.addField({
        id: 'refresh_link',
        type: 'inlinehtml',
        label: 'Refresh link'
    });
    refreshLink.defaultValu = "<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
    };
})

```

portlet.resize

Method Description	Resizes a form portlet type immediately.
Returns	Void

Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/portlet Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/portlet Module Script Sample](#).

```
//Add additional code
...
portlet.resize();
...
//Add additional code
```

portlet.refresh

Method Description	Refreshes a form portlet type immediately.
Returns	Void
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/portlet Module
Since	2016.1

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/portlet Module Script Sample](#).

```
...
portlet.refresh();
...
```

N/query Module

Load the query module to create and run queries using the SuiteAnalytics Workbook query engine. For more information, see the help topic [SuiteAnalytics Workbook Beta](#). Using the query module, you can:

- Use multilevel joins to create queries using field data from multiple record types.
- Create conditions (filters) using AND, OR, and NOT logic, as well as formulas.
- Sort query results based on the values of multiple columns.
- Load and delete existing saved queries that were created using the SuiteAnalytics Workbook UI.
- View paged query results.
- Use promises for asynchronous execution.



Important: The N/query module lets you create and run queries using the SuiteAnalytics Workbook query engine. In the 2018.2 release, you can use the N/query module to load and delete existing searches, but you cannot save searches. You can save searches using the SuiteAnalytics Workbook UI.

For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For information, see the help topic [Supported Record Types for the SuiteAnalytics Workbook Beta Period](#).

- [N/query Module Members](#)
- [Column Object Members](#)
- [Component Object Members](#)
- [Condition Object Members](#)
- [Page Object Members](#)
- [PagedData Object Members](#)
- [PageRange Object Members](#)
- [Query Object Members](#)
- [Result Object Members](#)
- [ResultSet Object Members](#)
- [Sort Object Members](#)
- [N/query Module Script Samples](#)

N/query Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	query.Column	Object	Client and server-side scripts	Encapsulates the field types (query result columns) that are displayed from the query results. Use Query.createColumn(options) or Component.createColumn(options) to create this object.
	query.Component	Object	Client and server-side scripts	Encapsulates one component of the query definition. The query definition always contains at least one component that encapsulates the initial search type. Queries with joins contain multiple components that encapsulate the join relationships. The initial component (Query.root) is automatically created with the query definition (query.Query). Use Query.autoJoin(options) or Component.autoJoin(options) to create subsequent components.
	query.Condition	Object	Client and server-side scripts	Encapsulates a condition. A condition narrows the query results. Use Query.createCondition(options) or Component.createCondition(options) to create this object.
	query.Page	Object	Client and server-side scripts	Encapsulates one page of the paged query results.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	query.PagedData	Object	Client and server-side scripts	Encapsulates a set of paged query results. This object also contains information about the set of paged results it encapsulates.
	query.PageRange	Object	Client and server-side scripts	Encapsulates a range of pages from the paged query results.
	query.Result	Object	Client and server-side scripts	Encapsulates a single row of the query result set.
	query.ResultSet	Object	Client and server-side scripts	Encapsulates the set of results returned by the query.
	query.Query	Object	Client and server-side scripts	Encapsulates the query definition. Use query.create(options) or query.load(options) to create this object. The creation of this object is the first step in creating a query with the N/query Module.
	query.Sort	Object	Client and server-side scripts	Encapsulates a sort that is placed on a particular query result column. Use Query.createSort(options) or Component.createSort(options) to create this object.
Method	query.create(options)	query.Query	Client and server-side scripts	Creates the query definition. The execution of this method is the first step in creating a query with the N/query Module.
	query.delete(options)	void	Client and server-side scripts	Deletes an existing query that was created using the SuiteAnalytics Workbook UI. The deleted query is no longer available and cannot be modified or executed.
	query.load(options)	query.Query	Client and server-side scripts	Loads an existing query that was created using the SuiteAnalytics Workbook UI. The loaded query can be modified (for example, by setting additional property values), joined with other search types, and executed in the same way as queries created using query.create(options) .
Enum	query.Aggregate	enum	Client and server-side scripts	Holds the string values for aggregate functions supported with the N/query Module . This enum is used to pass the aggregate function argument to Component.createColumn(options) , Component.createCondition(options) , Query.createColumn(options) , and Query.createCondition(options) .
	query.Operator	enum	Client and server-side scripts	Holds the string values for operators supported with the N/query Module . This enum is used to pass the operator argument to

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				Query.createCondition(options) and Component.createCondition(options) .
	query.ReturnType	enum	Client and server-side scripts	Holds the string values for the formula return types supported with the N/query Module . This enum is used to pass the formula return type argument to Query.createColumn(options) , Component.createColumn(options) , Query.createCondition(options) , and Component.createCondition(options) .
	query.SortLocale	enum	Client and server-side scripts	Holds the string values for sort locales supported with the N/query Module . This enum is used to pass the sort locale argument to Query.createSort(options) and Component.createSort(options) .
	query.Type	enum	Client and server-side scripts	Holds the string values for supported search types used in the query definition. This enum is used to pass the initial search type argument to query.create(options) .

Column Object Members

The following members are called on the [query.Column](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	Column.aggregate	string (read-only)	Client and server-side scripts	Describes an aggregate function that is performed on the query result column. An aggregate function performs a calculation on the column values and returns a single value.
	Column.component	query.Component (read-only)	Client and server-side scripts	Holds a reference to the query.Component object to which this query result column belongs.
	Column.fieldId	string (read-only)	Client and server-side scripts	Holds the name of the query result column. This property and the Column.formula property cannot be set at the same time.
	Column.formula	string (read-only)	Client and server-side scripts	Describes the formula used to create the query result column. This property and the Column.fieldId property cannot be set at the same time.
	Column.groupBy	boolean (read-only)	Client and server-side scripts	Indicates whether the query results are grouped by this query result column.
	Column.type	string (read-only)	Client and server-side scripts	Describes the return type of the formula used to create the query result column.

Component Object Members

The following members are called on the `query.Component` object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Method	<code>Component.autoJoin(options)</code>	<code>query.Component</code>	Client and server-side scripts	Creates a join relationship. After you create the initial query definition, use <code>Query.autoJoin(options)</code> to create your first join. Then use this method to create each subsequent join.
	<code>Component.createColumn(options)</code>	<code>query.Column</code>	Client and server-side scripts	Creates a query result column based on the component. Use this method to create columns based on the join relationships created with <code>Query.autoJoin(options)</code> and <code>Component.autoJoin(options)</code> .
	<code>Component.createCondition(options)</code>	<code>query.Condition</code>	Client and server-side scripts	Creates a condition (filter column) based on the component. Use this method to create conditions based on the join relationships created with <code>Query.autoJoin(options)</code> and <code>Component.autoJoin(options)</code> .
	<code>Component.createSort(options)</code>	<code>query.Sort</code>	Client and server-side scripts	Creates a sort based on the component. Use this method to create sorts based on the join relationships created with <code>Query.autoJoin(options)</code> and <code>Component.autoJoin(options)</code> .
	<code>Component.join(options)</code>	<code>query.Component</code>	Client and server-side scripts	Creates a join relationship. This method is an alias to <code>Component.autoJoin(options)</code> . After you create the initial query definition, use <code>Query.autoJoin(options)</code> to create your first join. Then use this method, or <code>Component.autoJoin(options)</code> , to create each subsequent join.
	<code>Component.joinFrom(options)</code>	<code>query.Component</code>	Client and server-side scripts	Creates an explicit directional join relationship from another component to this component (an inverse join). This method sets the <code>Component.source</code> property on the returned <code>query.Component</code> object. After you create the initial query definition, use this method to create explicit directional joins from other components to this component.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
	Component.joinTo(options)	query.Component	Client and server-side scripts	Creates an explicit directional join relationship to another component from this component (a polymorphic join). You can use this method to specify the target of the join when a field can join multiple search types. This method sets the Component.target property on the returned query.Component object. After you create the initial query definition, use this method to create explicit directional joins to other components from this component.
Property	Component.child	Object (read-only)	Client and server-side scripts	Describes the child components of the component. This property holds an object of key/value pairs. Each key is the name of a child component. Each value is the corresponding child query.Component object.
	Component.parent	string (read-only)	Client and server-side scripts	Describes the parent query.Component object of the component.
	Component.source	string (read-only)	Client and server-side scripts	Describes the source search type of the component. The value of this property is set when Component.joinFrom(options) is called to perform an explicit directional join from another component.
	Component.target	string (read-only)	Client and server-side scripts	Describes the target search type of the component. The value of this property is set when Component.joinTo(options) is called to perform an explicit directional join to another component.
	Component.type	string (read-only)	Client and server-side scripts	Describes the search type of the component.

Condition Object Members

The following members are called on the [query.Condition](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	Condition.aggregate	string (read-only)	Client and server-side scripts	Describes an aggregate function that is performed on the condition. An aggregate function performs a calculation on the condition values and returns a single value.
	Condition.children	query.Condition[] (read-only)	Client and server-side scripts	Holds an array of child conditions used to create the parent condition.
	Condition.component	query.Component (read-only)	Client and server-side scripts	Holds a reference to the query.Component object to which this condition belongs.
	Condition.fieldId	string (read-only)	Client and server-side scripts	Holds the name of the condition.
	Condition.formula	string (read-only)	Client and server-side scripts	Describes the formula used to create the condition.
	Condition.operator	string (read-only)	Client and server-side scripts	Holds the name of the operator used to create the condition.
	Condition.type	string (read-only)	Client and server-side scripts	The return type of the formula used to create the condition.
	Condition.values	string[] (read-only)	Client and server-side scripts	Holds an array of values used by an operator to create the condition.

Page Object Members

The following members are called on the [query.Page](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	Page.data	query.ResultSet (read-only)	Client and server-side scripts	References the query results contained in this page.
	Page.isFirst	boolean (read-only)	Client and server-side scripts	Indicates whether this page is the first of the paged query results.
	Page.isLast	boolean (read-only)	Client and server-side scripts	Indicates whether this page is the last of the paged query results.
	Page.pagedData	query.PagedData (read-only)	Client and server-side scripts	References the set of paged query results that this page is from.
	Page.pageRange	query.PageRange (read-only)	Client and server-side scripts	The range of query results for this page.

PagedData Object Members

The following members are called on the [query.PagedData](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Method	PagedData.iterator()	Iterator object	Client and server-side scripts	Standard SuiteScript 2.0 object for iterating through results.
Property	PagedData.count	number (read-only)	Client and server-side scripts	Describes the total number of paged query results.
	PagedData.pageRanges	query.PageRange[]	Client and server-side scripts	Holds an array of page ranges for the set of paged query results.
	PagedData.pageSize	number (read-only)	Client and server-side scripts	Describes the number of query result rows per page.

PageRange Object Members

The following members are called on the [query.PageRange](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	PageRange.index	number (read-only)	Client and server-side scripts	Describes the array index for this page range.
	PageRange.size	number (read-only)	Client and server-side scripts	Describes the number of query result rows in this page range.

Query Object Members

The following members are called on the [query.Query](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Method	Query.and()	query.Condition object	Client and server-side scripts	Creates a new condition (a query.Condition object) that corresponds to a logical conjunction (AND) of the arguments passed to the method. The arguments must be one or more query.Condition objects.
	Query.autojoin(options)	query.Component	Client and server-side scripts	Creates a join relationship. After you create the initial query definition, use this method to create your first join.
	Query.createColumn(options)	query.Column object	Client and server-side scripts	Creates a query result column based on the query.Query object. Use this method to create columns on the initial query

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
				definition created with query.create(options) .
	Query.createCondition(options)	query.Condition object	Client and server-side scripts	Creates a condition (filter column) based on the query.Query object. Use this method to create conditions on the initial query definition created with query.create(options) .
	Query.createSort(options)	query.Sort object	Client and server-side scripts	Creates a sort based on the query.Query object. The query.Sort object describes a sort that is placed on a particular query result column or condition.
	Query.join(options)	query.Component	Client and server-side scripts	Creates a join relationship. This method is an alias to Query.autojoin(options) . After you create the initial query definition, use this method, or Query.autojoin(options) , to create your first join.
	Query.joinFrom(options)	query.Component	Client and server-side scripts	Creates an explicit directional join relationship from another component to the root component of the search definition (an inverse join). This method sets the Component.source property on the returned query.Component object. After you create the initial query definition, use this method to create your first join as an explicit directional join from another component to this component.
	Query.joinTo(options)	query.Component	Client and server-side scripts	Creates an explicit directional join relationship to another component from this component (a polymorphic join). You can use this method to specify the target of the join when a field can join multiple search types. This method sets the Component.target property on the returned query.Component object. After you create the initial query definition, use this method to create your first join as an explicit directional join to another component from this component.
	Query.not()	query.Condition	Client and server-side scripts	Creates a new condition (a query.Condition object) that corresponds to a logical

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
				negation (NOT) of the argument passed to the method. The argument must be a query.Condition object.
	Query.or()	query.Condition	Client and server-side scripts	Creates a new condition (a query.Condition object) that corresponds to a logical disjunction (OR) of the arguments passed to the method. The arguments must be one or more query.Condition objects.
	Query.run()	query.ResultSet	Client and server-side scripts	Executes the query and returns the query result set.
	Query.run.promise()	query.ResultSet	Client scripts	Executes the query asynchronously and returns the query result set.
	Query.runPaged()	query.PagedData	Client and server-side scripts	Executes the query and returns a set of paged results.
	Query.runPaged.promise()	query.PagedData	Client scripts	Executes the query asynchronously and returns a set of paged results.
Property	Query.child	Object (read-only)	Client and server-side scripts	Holds a references to children of the root component of the query definition. The value of this property is an object of key/value pairs. Each key is the name of a child component. Each respective value is the corresponding query.Component object.
	Query.columns	query.Column[]	Client and server-side scripts	Holds an array of query result columns returned from the query. Before you execute the query, you must assign all created columns as array values to this property.
	Query.condition	query.Condition object	Client and server-side scripts	References the parent condition that narrows the query results. Before you execute the query, you must assign your simple or complex conditions to this property.
	Query.id	number (read-only)	Client and server-side scripts	Holds the ID of the query definition. This property has a value only for existing queries that are loaded using query.load(options) . If you create a query using

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
				query.create(options) but do not save it, this property is null.
	Query.name	string (read-only)	Client and server-side scripts	Holds the name of the query definition. This property has a value only for existing queries that are loaded using query.load(options) . If you create a query using query.create(options) but do not save it, this property is null.
	Query.root	query.Component (read-only)	Client and server-side scripts	References the root component of the query definition.
	Query.sort	query.Column[] (read-only)	Client and server-side scripts	Holds an array of query result columns used for sorting.
	Query.type	string (read-only)	Client and server-side scripts	Holds the search type of the initial query definition.

Result Object Members

The following members are called on the [query.Result](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	Result.columns	query.Column[] (read-only)	Client and server-side scripts	Holds an array of query result column references.
	Result.values	string[] or number[] or boolean[] (read-only)	Client and server-side scripts	Describes the result values.

ResultSet Object Members

The following members are called on the [query.ResultSet](#) object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Method	ResultSet.iterator()	Iterator object	Client and server-side scripts	Standard SuiteScript 2.0 object for iterating through results.
Property	ResultSet.columns	query.Column[] (read-only)	Client and server-side scripts	Holds an array of query result column references.
	ResultSet.results	query.Result[] (read-only)	Client and server-side scripts	Holds an array of query.Result objects.
	ResultSet.types	string[] (read-only)	Client and server-side scripts	Holds an array of the return types for ResultSet.results .

Sort Object Members

The following members are called on the `query.Sort` object.

Member Type	Name	Return Type/Value Type	Supported Script Types	Description
Property	<code>Sort.ascending</code>	boolean	Client and server-side scripts	Indicates whether the sort direction is ascending.
	<code>Sort.caseSensitive</code>	boolean	Client and server-side scripts	Indicates whether the sort is case sensitive. If a sort is case sensitive (and the sort direction is ascending), rows with column values that start with uppercase letters are listed before rows with column values that start with lowercase letters. If a sort is not case sensitive, uppercase and lowercase letters are treated the same.
	<code>Sort.column</code>	<code>query.Column</code> (read-only)	Client and server-side scripts	Describes the query result column that the query results are sorted by.
	<code>Sort.locale</code>	string	Client and server-side scripts	The locale to use for the sort. A locale represents a combination of language and region, and it can affect how certain values (such as strings) are sorted.
	<code>Sort.nullsLast</code>	boolean	Client and server-side scripts	Indicates whether query results with null values are listed at the end of the query results.

N/query Module Script Samples

```
require(['N/query'],
function(query) {
    var search = query.create({
        type: query.Type.CUSTOMER
    });

    var salesrep = search.autoJoin({
        fieldId: 'salesrep'
    });
    var location = salesrep.autoJoin({
        fieldId: 'location'
    });

    var cond1 = search.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 107
    });
    var cond2 = search.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 2647
    });
});
```

```

});
var cond3 = salesrep.createCondition({
  fieldId: 'email',
  operator: query.Operator.START_WITH_NOT,
  values: 'foo'
});
search.condition = search.and(
  cond3, search.or(cond1, cond2)
);

search.columns = [
  search.createColumn({
    fieldId: 'entityid'
  }),
  search.createColumn({
    fieldId: 'id'
  }),
  salesrep.createColumn({
    fieldId: 'entityid'
  }),
  salesrep.createColumn({
    fieldId: 'email'
  }),
  salesrep.createColumn({
    fieldId: 'hiredate'
  }),
  location.createColumn({
    fieldId: 'name'
  })
];
search.sort = [
  search.createSort({
    column: search.columns[3]
  }),
  search.createSort({
    column: search.columns[0],
    ascending: false
  })
];

var resultSet = search.run();

var results = resultSet.results;
for (var i = results.length - 1; i >= 0; i--)
  log.debug(results[i].values);
log.debug(resultSet.types);

log.error(
  search.root === location.parent.parent
);
log.error(
  search.root.child.salesrep === location.parent
);
log.error(
  search.child.salesrep === location.parent
);

```

```

    );
    log.error(
        search.child.salesrep.child.location === location
    );
});

```

```

require(['N/query'],
function(query) {
    var search = query.create({
        type: query.Type.TRANSACTION
    });

    var entity = search.autoJoin({
        fieldId: 'entity'
    });

    search.columns = [entity.createColumn({
        fieldId: 'subsidiary'
    })];

    search.sort = [search.createSort({
        column: search.columns[0],
        ascending: false
    })];

    var results = search.runPaged({
        pageSize: 10});

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

    // First way to fetch results
    var iterator = results.iterator();
    iterator.each(function(result) {
        var page = result.value;
        log.debug(page.pageRange.size);
        return true;
    })

    // Second way to fetch results
    for (var i = 0; i < results.pageRanges.length; i++) {
        var page = results.fetch(i);
        log.debug(page.pageRange.size);
    }

});

```

Scripting with the N/query Module

The N/query module lets you create and run queries using the SuiteAnalytics Workbook query engine. Before you start creating your queries, you should be familiar with the module objects and how to use them, as well as some of the terminology used in the N/query module. You can also take a look at a script walkthrough that explains how to create queries using different approaches.

■ [N/query Module Objects](#)

- [N/query Module Terminology](#)
- [N/query Module Script Walkthrough](#)

N/query Module Objects

The N/query module includes the following objects:

- [Query and Component Objects](#)
- [Condition Object](#)
- [Column Object](#)
- [Sort Object](#)
- [ResultSet and Result Objects](#)
- [Page, PagedData, and PageRange Objects](#)

Query and Component Objects

The [query.Query](#) object and the [query.Component](#) object are the primary building blocks for a query created with the N/query module. Each query creates one [query.Query](#) object and one or more [query.Component](#) objects. The [query.Query](#) object encapsulates the query definition, and the [query.Component](#) object encapsulates one component of the query definition.

To create a query with the N/query module:

1. Use the [query.create\(options\)](#) method to create your initial query definition (the [query.Query](#) object). The initial query definition uses one search type. For available search types, see [query.Type](#).
2. After you create the initial query definition, use [Query.autoJoin\(options\)](#), [Query.joinFrom\(options\)](#), or [Query.joinTo\(options\)](#) to create your first join.
3. Use [Component.autoJoin\(options\)](#), [Component.joinFrom\(options\)](#), or [Component.joinTo\(options\)](#) to create all subsequent joins.

The query definition always contains at least one [query.Component](#) object. Each new component is created as a child of the previous component, and all components exist as children of the query definition. You can think of a component as a building block; each new component builds on the previous component created. The last component created encapsulates the relationship between it and all of its parent components.

Queries with joins contain multiple components. The query definition contains a child [query.Component](#) object for each of the following:

- **The initial query definition:** The initial [query.Component](#) object is called the root component. It encapsulates the initial search type passed to [query.create\(options\)](#). The root component is automatically created with the initial query definition and is a child to the [query.Query](#) object. The [Query.root](#) property contains a reference to the root component.
- **The first join:** The second [query.Component](#) object is created with [Query.autoJoin\(options\)](#), [Query.joinFrom\(options\)](#), or [Query.joinTo\(options\)](#). It encapsulates the relationship between the initial query definition and the second search type. This relationship is determined by the join ID passed to these methods, as well as whether [Query.joinFrom\(options\)](#) or [Query.joinTo\(options\)](#) was used to create an explicit directional join. The second [query.Component](#) object is a child to the root component.
- **Each subsequent join:** The third [query.Component](#) object is created with [Component.autoJoin\(options\)](#), [Component.joinFrom\(options\)](#), or [Component.joinTo\(options\)](#). All subsequent joins are also created using these methods. Each of these [query.Component](#) objects encapsulates the relationship between all previous search types and the new search

type. This relationship is determined by the join ID passed to these methods, as well as whether `Component.joinFrom(options)` or `Component.joinTo(options)` was used to create an explicit directional join.

Condition Object

A condition narrows the query results. The `query.Condition` object performs the same function as the `search.Filter` object in the [N/search Module](#). The primary difference is that `query.Condition` objects can contain other `query.Condition` objects.

To create conditions:

- Use `Query.createCondition(options)` to create conditions for the initial query definition created with `query.create(options)`.
- Use `Component.createCondition(options)` to create conditions for the join relationships created with `Query.autoJoin(options)`, `Query.joinFrom(options)/Query.joinTo(options)`, `Component.autoJoin(options)`, or `Component.joinFrom(options)/Component.joinTo(options)`.
- If you have multiple conditions, use `Query.and()`, `Query.or()`, and `Query.not()` to create a new nested condition.
- If you want to use a formula to define your conditions, assign the formula to `Condition.formula`.
- Assign your simple or nested conditions as array values to `Query.condition`.

Column Object

The `query.Column` object is the equivalent of the `search.Column` object in the [N/search Module](#). The `query.Column` object describes the field types (columns) that are displayed from the query results.

To create columns:

- Use `Query.createColumn(options)` to create a column on the initial query definition created with `query.create(options)`.
- Use `Component.createColumn(options)` to create a column on a join relationship created with `Query.autoJoin(options)`, `Query.joinFrom(options)/Query.joinTo(options)`, `Component.autoJoin(options)`, or `Component.joinFrom(options)/Component.joinTo(options)`.
- If you want to use a formula to define your columns, assign the formula to `Column.formula`.
- Assign all created columns as array values to `Query.columns`.

Sort Object

The `query.Sort` object describes how query results are sorted (for example, ascending or descending, case sensitive or case insensitive, and so on).

To create a sort:

- Use `Query.createSort(options)` to create a sort on the initial query definition created with `query.create(options)`.
- Use `Component.createSort(options)` to create a sort based on a join relationship created with `Query.autoJoin(options)`, `Query.joinFrom(options)/Query.joinTo(options)`, `Component.autoJoin(options)`, or `Component.joinFrom(options)/Component.joinTo(options)`.
- Assign all created sorts as array values to `Query.sort`.

ResultSet and Result Objects

When you are ready to execute your query, call `Query.run()`. This method returns a `query.ResultSet` object, which encapsulates the metadata for the set of results returned by the query.

To access your actual query results, iterate through the `ResultSet.results` array. Each member of the `ResultSet.results` array is a `query.Result` object. The `query.Result` object encapsulates a single row of the result set.

Page, PagedData, and PageRange Objects

You also can execute your query by calling `Query.runPaged()`. This method returns a `query.PagedData` object, which encapsulates a set of paged query results.

To access your query results, iterate through the paged query results using `PagedData.iterator()`. You can access each page of the query results, which are represented by `query.Page` objects. The `query.PageRange` object encapsulates the range of query results for a page.

N/query Module Terminology

Term	Definition	For More Information
Aggregate function	An aggregate function performs a calculation on a column of values and returns a single value. You can add aggregate functions to conditions and query results columns.	See <code>query.Aggregate</code> , <code>Component.createColumn(options)</code> , <code>Component.createCondition(options)</code> , <code>Query.createColumn(options)</code> , and <code>Query.createCondition(options)</code> .
Column	A column describes the field types (columns) that are displayed from the query results. A column is also known as a query results column.	See <code>query.Column</code> .
Component	<p>When you script queries with the N/query module, your query is made up of one or more components, which are represented as <code>query.Component</code> objects. You can think of a component as a building block; each new component builds on the previous component created.</p> <ul style="list-style-type: none"> ■ The first component created represents the initial search type and is a child of <code>query.Query</code>. ■ Each subsequent component created is a child of the previous component. ■ The last component created encapsulates the join relationship between it and all of its parent components. <p>A query always contains at least one component: the root component. When you create the initial query definition using <code>query.create(options)</code>, the root component is created automatically. Queries with joins contain multiple components. A new component is created each time you create a join using one of the following methods:</p> <ul style="list-style-type: none"> ■ <code>Query.autoJoin(options)</code>, <code>Query.joinFrom(options)</code>, or <code>Query.joinTo(options)</code> ■ <code>Component.autoJoin(options)</code>, <code>Component.joinFrom(options)</code>, or <code>Component.joinTo(options)</code> 	See <code>query.Component</code> .
Condition	A condition narrows the query results.	See <code>query.Condition</code> .

Term	Definition	For More Information
Formula	Formulas can be used to create conditions and columns.	See the help topics SuiteAnalytics Workbook Beta , SQL Expressions , and Search Formula Examples and Tips .
Group	You can summarize your query results into unique groups of column values.	See Column.groupBy .
Join	A join lets you create a query based on a field type that is shared between two record types. You can use Query.autoJoin(options) and Component.autoJoin(options) to create a join relationship automatically based on a field that you specify. You can use Query.joinFrom(options)/Query.joinTo(options) and Component.joinFrom(options)/Component.joinTo(options) to create explicit directional join relationships from one component to another.	See query.Query and query.Component .
Page	A page represents one page from a set of paged query results. When you create a query with the N/query module, you can return the results as one result set or a set of paged results.	See Query.runPaged() , query.PagedData , query.PageRange , and query.Page .
Paged data	Paged data represents a set of paged query results.	See Query.runPaged() , query.PagedData , query.PageRange , and query.Page .
Page range	A page range is a set of pages from a set of paged query results.	See Query.runPaged() , query.PagedData , query.PageRange , and query.Page .
Result	A result is a single row from a result set.	See Query.run() , query.ResultSet and query.Result .
Result set	A result set is a set of query results.	See Query.run() , query.ResultSet and query.Result .
Query definition	The query definition is the initial search type you define, plus any subsequent joins you define. The initial query definition is created with query.create(options) .	See query.Query .
Search type	The search type is the initial search type of your query definition. It represents the record type you want to search for. It is set with the query.Type enum during the execution of query.create(options) . For example, if you want to search for customer records, specify query.Type.CUSTOMER as the search type when you call query.create(options) .	See query.Query and query.Type .
Sort	A sort is placed on a query results column to describe how the query results are sorted (for example, ascending or descending, case sensitive or case insensitive, and so on).	See query.Sort , Query.createSort(options) , and Component.createSort(options) .

N/query Module Script Walkthrough

This topic walks through the two script examples shown under [N/query Module Script Samples](#).

Example 1

```
require(['N/query'],
```

```

function(query) {

    // Use query.create(options) to create your initial
    // query definition.
    var search = query.create({
        type: query.Type.CUSTOMER
    });

    // Use Query.autoJoin(options) to create your first join.
    var salesrep = search.autoJoin({
        fieldId: 'salesrep'
    });

    // Use Component.autoJoin(options) to create your second
    // join and each subsequent join.
    var location = salesrep.autoJoin({
        fieldId: 'location'
    });

    // Use Query.createCondition(options) to create
    // conditions for your initial query definition.
    var cond1 = search.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 107
    });
    var cond2 = search.createCondition({
        fieldId: 'id',
        operator: query.Operator.EQUAL,
        values: 2647
    });

    // Use Component.createCondition(options) to create
    // conditions for your joins
    var cond3 = salesrep.createCondition({
        fieldId: 'email',
        operator: query.Operator.START_WITH_NOT,
        values: 'foo'
    });

    // If you have one condition, assign it to the
    // Query.condition property.
    // If you have multiple conditions, logically
    // connect them with Query.and(), Query.or(),
    // and Query.not(). Then assign the statement to the
    // Query.condition property.
    search.condition = search.and(
        cond3, search.or(cond1, cond2)
    );

    // Use Query.createColumn(options) to create columns
    // for your initial query definition. Use
    // Component.createColumn(options) to create columns for
    // your joins. Assign each column, as an array member, to
    // the Query.columns property.

```

```

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
    location.createColumn({
        fieldId: 'name'
    })
];

// Use Query.createSort(options) to create an ascending or
// descending sort on columns created for your initial
// query definition. Assign each sort, as an array member,
// to the Query.sort property.
search.sort = [
    search.createSort({
        column: search.columns[3]
    }),
    search.createSort({
        column: search.columns[0],
        ascending: false
    })
];

// Use Query.run() to synchronously execute your query
// and return the metadata for a set of results. You can use
// Query.promise.run() as an asynchronous alternative.
var resultSet = search.run();

// The ResultSet.results property holds an array of your actual
// results. Each array member is a query.Result object. Iterate
// through the array to access the results.
var results = resultSet.results;
results.forEach(function(result) {
    log.debug(result.values);
});
log.debug(resultSet.types);

log.error(
    search.root === location.parent.parent
);
log.error(
    search.root.child.salesrep === location.parent
);

```

```

    log.error(
      search.child.salesrep === location.parent
    );
    log.error(
      search.child.salesrep.child.location === location
    );
  });

```

Example 2

```

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

    // Use query.create(options) to create your initial
    // query definition.
    var search = query.create({
      type: query.Type.TRANSACTION
    });

    // Use query.autoJoin(options) to create your first join.
    var entity = search.autoJoin({
      fieldId: 'entity'
    });

    // Use Query.createColumn(options) to create columns
    // for your initial query definition. Use
    // Component.createColumn(options) to create columns for
    // your joins. Assign each column, as an array member, to
    // the Query.columns property.
    search.columns = [
      entity.createColumn({
        fieldId: 'subsidiary'
      })
    ];

    // Use Query.createSort(options) to create an ascending or
    // descending sort on columns created for your initial
    // query definition. Assign each sort, as an array member,
    // to the Query.sort property.
    search.sort = [
      search.createSort({
        column: search.columns[0],
        ascending: false
      })
    ];

    // Use Query.runPaged() to synchronously execute your query
    // and return the metadata for an array of paged results. You can use
    // Query.promise.runPaged() as an asynchronous alternative.
    var results = search.runPaged({
      pageSize: 10
    });

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

```

```

// Use one of the following ways to iterate through the array
// to access the paged results.

// First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
});

// Second way to fetch results (you can also use a forEach loop)
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}
});

```

query.Column

Object Description	<p>Encapsulates a query result column.</p> <p>The <code>query.Column</code> object is the equivalent of the search.Column object in the N/search Module. The <code>query.Column</code> object describes the field types (columns) that are displayed from the query results.</p> <p>To create columns:</p> <ul style="list-style-type: none"> ■ Use Query.createColumn(options) to create a column on the initial query definition created with query.create(options). ■ Use Component.createColumn(options) to create a column on a join relationship created with Query.autoJoin(options) or Component.autoJoin(options). ■ Assign all created columns as array values to Query.columns. For an example, see Syntax.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Column Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```

var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

```

```

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();

```

Column.aggregate

Property Description	Describes an aggregate function that is performed on the query result column. An aggregate function performs a calculation on the column values and returns a single value. This property is set when Query.createColumn(options) or Component.createColumn(options) is executed. For a list of supported aggregate functions, see the query.Aggregate enum.
Type	string (read-only)
Module	N/query Module
Parent Object	query.Column
Sibling Object Members	Column Object Members
Since	2018.1

Column.component

Property Description	Holds a reference to the query.Component object to which this query result column belongs. This property is set when Query.createColumn(options) or Component.createColumn(options) is executed.
-----------------------------	--

Type	query.Component object (read-only)
Module	N/query Module
Parent Object	query.Column
Sibling Object Members	Column Object Members
Since	2018.1

Column.fieldId

Property Description	Holds the name of the query result column. This property is set during the execution of Query.createColumn(options) or Component.createColumn(options) . This property and the Column.formula property cannot be set at the same time.
Type	string (read-only)
Module	N/query Module
Parent Object	query.Column
Sibling Object Members	Column Object Members
Since	2018.1

Column.formula

Property Description	Describes a formula used to create the query result column. This property is set during the execution of Query.createColumn(options) or Component.createColumn(options) . This property and the Column.fieldId property cannot be set at the same time. For more information on formulas, see the help topics SuiteAnalytics Workbook Beta , SQL Expressions , and Search Formula Examples and Tips .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Column
Sibling Object Members	Column Object Members
Since	2018.1

Column.groupBy

Property Description	Indicates whether the query results are grouped by this query result column. This property is set during the execution of Component.createColumn(options) .
Type	boolean (read-only)
Module	N/query Module
Parent Object	query.Column

Sibling Object Members	Column Object Members
Since	2018.1

Column.type

Property Description	Describes the return type of the formula used to create the query result column. This property is set during the execution of Query.createColumn(options) or Component.createColumn(options) . If a formula is specified when these methods are called, this property contains the return type of the formula. If a formula is not specified, this property is null. For more information on formulas, see the help topics SuiteAnalytics Workbook Beta , SQL Expressions , and Search Formula Examples and Tips .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Column
Sibling Object Members	Column Object Members
Since	2018.1

query.Component

Object Description	<p>Encapsulates one component of the query definition. Each new component is created as a child to the previous component. All components exist as children to the query definition (query.Query). You can think of a component as a building block; each new component builds on the previous component created. The last component created encapsulates the relationship between it and all of its parent components.</p> <p>The query definition always contains at least one component. Queries with joins contain multiple components. The query definition (query.Query) contains a child <code>query.Component</code> object for each of the following:</p> <ul style="list-style-type: none"> ■ The initial query definition: The initial <code>query.Component</code> object is called the root component. It encapsulates the initial search type passed to query.create(options). The root component is automatically created with the query.Query object and is a child of the query.Query object. The Query.root property contains a reference to the root component. ■ The first join: The second <code>query.Component</code> object is created with Query.autoJoin(options). It encapsulates the relationship between the initial query definition and the second search type. This relationship is determined by the join ID passed to Query.autoJoin(options). The second <code>query.Component</code> object is a child of the root component. ■ Each subsequent join: The third <code>query.Component</code> object is created with Component.autoJoin(options). All subsequent joins and their respective <code>query.Component</code> objects are also created with Component.autoJoin(options). Each of these <code>query.Component</code> objects encapsulates the relationship between all previous search types and the new search type. This relationship is determined by the join ID passed to Component.autoJoin(options).
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Component Object Members

Since	2018.1
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();
```

Component.autoJoin(options)

Method Description	Creates a join relationship. Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type . After you create the initial query definition, use Query.autoJoin(options) to create your first join (query.Component). Then use <code>Component.autoJoin(options)</code> to create each subsequent join (query.Component).
---------------------------	---

	 Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period .
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	<p>The column type (field type) that joins the parent component to the new component. Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.autoJoin({
    fieldId: 'entity'
});
```

```

search.columns = [entity.createColumn({
    fieldId: 'subsidiary'
})];

search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
})];

var results = search.runPaged({
    pageSize: 10
});

```

Component.createColumn(options)

Method Description	<p>Creates a query result column based on the query.Component object. The query.Column object is the equivalent of the search.Column object in the N/search Module. The query.Column object describes the field types (columns) that are displayed from the query results.</p> <p>To create columns:</p> <ul style="list-style-type: none"> ■ Use <code>Component.createColumn(options)</code> to create conditions on the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). Use this method in one of two ways: <ul style="list-style-type: none"> □ Pass in an argument for the parameter <code>options.fieldId</code>. □ Pass in an argument for the parameter <code>options.formula</code>. If you use this option, you can also use the optional parameter <code>options.type</code>. ■ If needed, use Query.createColumn(options) to create columns on the initial query definition created with query.create(options). ■ Assign all created columns as array values to Query.columns. For an example, see Syntax.
Returns	query.Column object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.fieldId</code>	string	required if <code>options.formula</code> is not used	The name of the query result column. This value sets the Column.fieldId property. Obtain this value from the Records Browser :

Parameter	Type	Required / Optional	Description
			<ol style="list-style-type: none"> 1. Go to the appropriate record type. 2. Scroll until you see the Search Columns table. 3. Locate the appropriate value in the Internal ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
options.formula	string	required if options.fieldId is not used	<p>The formula used to create the query result column. This value sets the Column.formula property.</p> <p>For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p>
options.type	string	optional if options.formula is used	<p>If you use the <code>options.formula</code> parameter, use this parameter to explicitly define the formula's return type. Defining the formula's return type might be required if the return type cannot be determined correctly based on the specified formula. This value sets the Column.type property.</p> <p>Use the appropriate query.ReturnType enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>
options.aggregate	string	optional	<p>Use this parameter to run an aggregate function on your query result column. An aggregate function performs a calculation on the column values and returns a single value. This value sets the Column.aggregate property.</p> <p>Use the appropriate query.Aggregate enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>
options.groupBy	boolean	optional	<p>Indicates whether the query results are grouped by this query result column. This value sets the Column.groupBy property.</p> <p>If you do not pass in an argument, the default value is set to <code>false</code>.</p>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    })
];
```

```

salesrep.createColumn({
    fieldId: 'entityid'
}),
salesrep.createColumn({
    fieldId: 'email'
}),
salesrep.createColumn({
    fieldId: 'hiredate'
}),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();

```

Component.createCondition(options)

Method Description	<p>Creates a condition (query filter) based on the query.Component object. A condition narrows the query results. The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p> <ul style="list-style-type: none"> ■ Use Component.createCondition(options) to create conditions on the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). Use this method in one of two ways: <ul style="list-style-type: none"> □ Pass in arguments for the parameters <code>options.fieldId</code>, <code>options.operator</code>, and <code>options.values</code>. The combination of these arguments translates to <i><filter column><operator><field value></i> (for example, 'city' equals 'Boston'). □ Pass in an argument for the parameter <code>options.formula</code>. If you use this option, you can also use the optional parameter <code>options.type</code>. ■ If needed, use Query.createCondition(options) to create conditions on the initial query definition created with query.create(options). ■ If you have multiple conditions, use them to create a new nested condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your simple or nested condition to Query.condition. For an example, see Syntax.
Returns	query.Condition object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component

Sibling Object Members	Component Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required if options.operator and options.values are used	<p>The name of the condition. This value sets the Condition.fieldId property. Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the appropriate record type. 2. Scroll until you see the Search Filters table. 3. Locate the appropriate value in the Internal ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
options.operator	string	required if options.fieldId and options.values are used	<p>The operator used by the condition. This value sets the Condition.operator parameter. Use the appropriate query.Operator enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>
options.values	string[]	required if options.fieldId and options.operator are used	<p>An array of string values. This value sets the Condition.values property.</p>
options.formula	string	required if options.fieldId, options.operator, and options.values are not used	<p>The formula used to create the condition. This value sets the Condition.formula property. For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p>
options.type	string	optional if options.formula is used	<p>If you use the <code>options.formula</code> parameter, use this parameter to explicitly define the formula's return type. Defining the formula's return type might be required if the return type cannot be determined correctly based on the specified formula. This value sets the Condition.type property. Use the appropriate query.ReturnType enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>
options.aggregate	string	optional	<p>Use this parameter to run an aggregate function on a condition. An aggregate function performs a calculation on the condition values and returns a single value. This value sets the Condition.aggregate property. Use the appropriate query.Aggregate enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});
var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
});

search.condition = search.and(
    cond3, search.not(
        search.or(cond1, cond2)
    )
);

var resultSet = search.run();
```

Component.createSort(options)

Method Description	<p>Creates a sort based on the query.Component object. The query.Sort object describes a sort that is placed on a particular query result column or condition.</p> <p>To create a sort:</p> <ul style="list-style-type: none"> ■ Use <code>Component.createSort(options)</code> to create a sort based on a join relationship created with Query.autoJoin(options) or Component.autoJoin(options). ■ Use Query.createSort(options) to create a sort based on the initial query definition created with query.create(options). ■ Assign all created sorts as array values to Query.sort. For an example, see Syntax.
Returns	query.Sort

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.column	query.Column	required	The query result column that you want to sort by. This value sets the Sort.column property.
options.ascending	boolean	optional	Indicates whether the sort direction is ascending. This value sets the Sort.ascending property. The default value of this property is <code>true</code> , meaning that the sort direction is ascending. If you want the sort direction to be descending, set this property to <code>false</code> .
options.caseSensitive	boolean	optional	Indicates whether the sort is case sensitive. This value sets the Sort.caseSensitive property. If a sort is case sensitive (and the sort direction is ascending), rows with column values that start with uppercase letters are listed before rows with column values that start with lowercase letters. If a sort is not case sensitive, uppercase and lowercase letters are treated the same. For example, the following list of items is sorted using a case-sensitive sort with a sort direction of ascending: <ul style="list-style-type: none"> Banana Orange apple grapefruit kiwi Here is the same list of items sorted using a regular (not case-sensitive) sort with a sort direction of ascending: <ul style="list-style-type: none"> apple Banana grapefruit kiwi Orange The default value of this property is <code>false</code> .
options.locale	string	optional	The locale to use for the sort. This value sets the Sort.locale property. A locale represents a combination of language and region, and it can affect how certain values (such as strings) are sorted. For example, languages that share

Parameter	Type	Required / Optional	Description
			the same alphabet may sort characters differently. Use this property to ensure that query results are sorted using locale-specific rules. Use the appropriate query.SortLocale enum value to pass in your argument. This enum holds all the supported values for this parameter.
options.nullsLast	boolean	optional	Indicates whether query results with null values are listed at the end of the query results. This value sets the Sort.nullsLast property. The default value of this property is the value of the options.ascending property. For example, if the options.ascending property is set to <code>true</code> , the options.nullsLast property is also set to <code>true</code> .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];
```

```
};

var resultSet = search.run();
```

Component.join(options)

Method Description	<p>Creates a join relationship. This method is an alias to Component.autoJoin(options). Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type. After you create the initial query definition, use Query.autoJoin(options) to create your first join (query.Component). Then use <code>Component.join(options)</code> to create each subsequent join (query.Component).</p> <div style="border: 1px solid #f1c40f; padding: 10px; margin-top: 10px;"> <p> Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	<p>The column type (field type) that joins the parent component to the new component. This value determines the columns on which the components are joined and the type of the newly joined component. Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
```

```

    type: query.Type.TRANSACTION
  });

  var entity = search.join({
    fieldId: 'entity'
  });

  search.columns = [entity.createColumn({
    fieldId: 'subsidiary'
  })];

  search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
  })];

  var results = search.runPaged({
    pageSize: 10
  });

```

Component.joinFrom(options)

Method Description	<p>Creates an explicit directional join relationship from another component to this component (an inverse join). This method sets the Component.source property on the returned query.Component object.</p> <p>Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type.</p> <p>After you create the initial query definition, use this method to create explicit directional joins from other components to this component.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"> <p> Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	<p>The column type (field type) that joins the parent component to the new component. Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
options.source	string	required	<p>The search type of the component joined to this component. This value sets the Component.source property. This value can be described as the inverse relationship of this component, and it determines the source search type of the newly joined component.</p>

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.EMPLOYEE
});

var salesorder = search.joinFrom({
    fieldId: 'salesrep',
    source: 'salesorder'
});

var items = salesorder.autoJoin({
    fieldId: 'item'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'hiredate'
    }),
    salesorder.createColumn({
        fieldId: 'id'
    })
];
```

```

    }},
    salesorder.createColumn({
        fieldId: 'trandate'
    })
  ];

  var sort1 = search.createSort({
    column: search.columns[0],
    ascending:false
  });
  var sort2 = search.createSort({
    column: search.columns[1],
    ascending:true
  });
  search.sort = [sort1, sort2];

  var results = search.run();

```

Component.joinTo(options)

Method Description	<p>Creates an explicit directional join relationship to another component from this component (a polymorphic join). This method sets the Component.target property on the returned query.Component object.</p> <p>Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type.</p> <p>After you create the initial query definition, use this method to create explicit directional joins to other components from this component.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"> <p> Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	The column type (field type) that joins the parent component to the new component.

Parameter	Type	Required / Optional	Description
			<p>Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
options.target	string	required	<p>The search type of the component joined to this component. This value sets the Component.target property.</p> <p>This value can be described as the polymorphic relationship of this component, and it determines the target search type of the newly joined component.</p>

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.joinTo({
    fieldId: 'entity',
    target: query.Type.CUSTOMER
});

search.columns = [
    entity.createColumn({
        fieldId: 'subsidiary'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false
    })
];

var results = search.runPaged({
    pageSize: 10
});
```


Component.child

Property Description	Holds a references to children of this component. The value of this property is an object of key/value pairs. Each key is the name of a child component. Each respective value refers to the corresponding query.Component object. The object values are set during the execution of Query.autoJoin(options) and Component.autoJoin(options) . The order of the key/value pairs reflects the parent/child hierarchy.
Type	Object (read-only)
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Component.parent

Property Description	Holds a references to the parent query.Component object of this component. This property is set during the execution of Query.autoJoin(options) or Component.autoJoin(options) .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Component.source

Property Description	Describes the search type of the component joined to this component. This property can also be described as the inverse relationship of this component. This property is set during the execution of Query.joinFrom(options) and Component.joinFrom(options) .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Component.target

Property Description	Describes the search type of this component. This property can also be described as the polymorphic relationship of this component.
-----------------------------	---

	This property is set during the execution of Query.joinTo(options) and Component.joinTo(options) .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

Component.type

Property Description	Describes the search type of this component. This property is set during the execution of Query.autoJoin(options) and Component.autoJoin(options) .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Component
Sibling Object Members	Component Object Members
Since	2018.1

query.Condition

Object Description	<p>A condition narrows the query results. The <code>query.Condition</code> object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that <code>query.Condition</code> objects can contain other <code>query.Condition</code> objects.</p> <p>To create conditions:</p> <ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions for the initial query definition created with query.create(options). ■ Use Component.createCondition(options) to create conditions for the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ If you have multiple conditions, use them to create a new nested condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your simple or nested condition to Query.condition. For an example, see Syntax.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Condition Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});
var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
});

search.condition = search.and(
    cond3, search.not(
        search.or(cond1, cond2)
    )
);

var resultSet = search.run();
```

Condition.aggregate

Property Description	<p>Describes an aggregate function that is performed on the condition. An aggregate function performs a calculation on the condition values and returns a single value. This property is set during the execution of <code>Query.createCondition(options)</code> or <code>Component.createCondition(options)</code>.</p> <div>  Note: This property is not applicable to parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	string (read-only)
Module	N/query Module

Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.children

Property Description	<p>Holds an array of child conditions used to create the parent condition.</p> <div>  Note: This property is applicable to only parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	query.Condition[]
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.component

Property Description	<p>Describes the component used to created the condition This property is set during the execution of <code>Query.createCondition(options)</code> and <code>Component.createCondition(options)</code>.</p> <div>  Note: This property is not applicable to parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	string (read-only)
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.fieldId

Property Description	<p>Holds the name of the condition. This property is set during the execution of <code>Query.createCondition(options)</code> and <code>Component.createCondition(options)</code>.</p> <div>  Note: This property is not applicable to parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	string (read-only)

Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.formula

Property Description	<p>Describes the formula used to create the condition. This property is set during the execution of Query.createCondition(options) and Component.createCondition(options). For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p> <div>  Note: This property is not applicable to parent conditions created with the execution of Query.and(), Query.or(), or Query.not(). </div>
Type	string (read-only)
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.operator

Property Description	<p>Holds the name of the operator used to create the condition. This property is set during the execution of Query.createCondition(options) and Component.createCondition(options).</p> <div>  Note: This property is not applicable to parent conditions created with the execution of Query.and(), Query.or(), or Query.not(). </div>
Type	string (read-only)
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.type

Property Description	<p>The return type of the formula used to create the condition. This property is set during the execution of Query.createCondition(options) or Component.createCondition(options).</p>
----------------------	--

	<p>For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p> <div>  Note: This property is not applicable to parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	string (read-only)
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

Condition.values

Property Description	<p>Holds an array of values used by an operator to create the condition. This property is set by passing in values for <code>options.fieldId</code>, <code>options.operator</code> and <code>options.values</code> during the execution of Query.createCondition(options) or Component.createCondition(options).</p> <div>  Note: This property is not applicable to parent conditions created with the execution of <code>Query.and()</code>, <code>Query.or()</code>, or <code>Query.not()</code>. </div>
Type	string[] (read-only)
Module	N/query Module
Parent Object	query.Condition
Sibling Object Members	Condition Object Members
Since	2018.1

query.Page

Object Description	One page of the paged query results.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Page Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var results = search.runPaged({
```

```

    pageSize: 10});

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

//First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
})

//Second way to fetch results
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}

```

Page.data

Property Description	References the query results contained in this page.
Type	query.ResultSet (read-only)
Module	N/query Module
Parent Object	query.Page
Sibling Object Members	Page Object Members
Since	2018.1

Page.isFirst

Property Description	Indicates whether the page is the first of the paged query results.
Type	boolean (read-only)
Module	N/query Module
Parent Object	query.Page
Sibling Object Members	Page Object Members
Since	2018.1

Page.isLast

Property Description	Indicates whether the page is the last of the paged query results.
Type	boolean (read-only)
Module	N/query Module
Parent Object	query.Page

Sibling Object Members	Page Object Members
Since	2018.1

Page.pageRange

Property Description	The range of query results for this page.
Type	query.PageRange (read-only)
Module	N/query Module
Parent Object	query.Page
Sibling Object Members	Page Object Members
Since	2018.1

Page.pagedData

Property Description	References the set of paged query results that this page is from.
Type	query.PagedData (read-only)
Module	N/query Module
Parent Object	query.Page
Sibling Object Members	Page Object Members
Since	2018.1

query.PagedData

Object Description	Encapsulates a set of paged query results. This object also contains information about the set of paged results it encapsulates. Use Query.runPaged() or Query.runPaged.promise() to create this object.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	PagedData Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var results = search.runPaged({
    pageSize: 10});
```



```

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

//First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
})

//Second way to fetch results
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}

```

PagedData.iterator()

Method Description	Standard SuiteScript 2.0 object for iterating through results
Returns	Iterator object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.PagedData
Sibling Object Members	PagedData Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```

var results = search.runPaged({
    pageSize: 10});

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

//First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
})

//Second way to fetch results

```

```
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}
```

PagedData.count

Property Description	Describes the total number of paged query result rows.
Type	number (read-only)
Module	N/query Module
Parent Object	query.PagedData
Sibling Object Members	PagedData Object Members
Since	2018.1

PagedData.pageRanges

Property Description	Holds an array of page ranges for the paged query results.
Type	query.PageRange[]
Module	N/query Module
Parent Object	query.PagedData
Sibling Object Members	PagedData Object Members
Since	2018.1

PagedData.pageSize

Property Description	Describes the number of query result rows per page.
Type	number (read-only)
Module	N/query Module
Parent Object	query.PagedData
Sibling Object Members	PagedData Object Members
Since	2018.1

query.PageRange

Object Description	Encapsulates the range of query results for a page.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	PageRange Object Members

Since	2018.1
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var results = search.runPaged({
    pageSize: 10});

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

//First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
})

//Second way to fetch results
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}
```

PageRange.index

Property Description	Describes the array index for this page range.
Type	number (read-only)
Module	N/query Module
Parent Object	query.PageRange
Sibling Object Members	PageRange Object Members
Since	2018.1

PageRange.size

Property Description	Describes the number of query result rows in this page range.
Type	number (read-only)
Module	N/query Module
Parent Object	query.PageRange
Sibling Object Members	PageRange Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var results = search.runPaged({
    pageSize: 10});

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

//First way to fetch results
var iterator = results.iterator();
iterator.each(function(result) {
    var page = result.value;
    log.debug(page.pageRange.size);
    return true;
})

//Second way to fetch results
for (var i = 0; i < results.pageRanges.length; i++) {
    var page = results.fetch(i);
    log.debug(page.pageRange.size);
}
```

query.Query

Object Description	<p>The <code>query.Query</code> object encapsulates the query definition. To create a query with the N/query module:</p> <ol style="list-style-type: none"> 1. Use the <code>query.create(options)</code> method to create your query definition (this object). The initial query definition uses one search type. For available search types, see query.Type. 2. After you create the initial query definition, use <code>Query.autoJoin(options)</code> to create your first join. 3. Then use <code>Component.autoJoin(options)</code> to create all subsequent joins. <p>The query definition always contains at least one <code>query.Component</code> object. The <code>query.Component</code> object encapsulates one component of the query definition. Each new component is created as a child to the previous component, and all components exist as children to the query definition. You can think of a component as a building block; each new component builds on the previous component created. The last component created encapsulates the relationship between it and all of its parent components.</p> <p>Queries with joins contain multiple components. The query definition contains a child <code>query.Component</code> object for each of the following:</p> <ul style="list-style-type: none"> ■ The initial query definition: The initial <code>query.Component</code> object is called the root component. It encapsulates the initial search type passed to <code>query.create(options)</code>. The root component is automatically created with the initial query definition and is a child to the <code>query.Query</code> object. The <code>Query.root</code> property contains a reference to the root component. ■ The first join: The second <code>query.Component</code> object is created with <code>Query.autoJoin(options)</code>. It encapsulates the relationship between the initial query definition and the second search type. This relationship is determined by the join ID passed to <code>Query.autoJoin(options)</code>. The second <code>query.Component</code> object is a child to the root component. ■ Each subsequent join: The third <code>query.Component</code> object is created with <code>Component.autoJoin(options)</code>. All subsequent joins are also created with
---------------------------	--

	Component.autoJoin(options) . Each of these query.Component objects encapsulates the relationship between all previous search types and the new search type. This relationship is determined by the join ID passed to Component.autoJoin(options) .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Query Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.join({
    fieldId: 'entity'
});

search.columns = [entity.createColumn({
    fieldId: 'subsidiary'
})];

search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
})];

var results = search.runPaged({
    pageSize: 10
});
```

Query.and()

Method Description	<p>Creates a new condition (a query.Condition object) that corresponds to a logical conjunction (AND) of the arguments passed to the method. The arguments must be one or more query.Condition objects.</p> <p>A condition narrows the query results. The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p> <ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions for the initial query definition created with query.create(options). ■ Use Component.createCondition(options) to create conditions for the join relationships created with Query.autoJoin(options) and Component.autoJoin(options).
---------------------------	--

	<ul style="list-style-type: none"> ■ If you have multiple conditions, use them to create a new parent condition with the methods <code>Query.and()</code>, <code>Query.or()</code>, and <code>Query.not()</code>. ■ Assign your parent condition to <code>Query.condition</code>. For an example, see Syntax.
Returns	<code>query.Condition</code> object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	<code>query.Query</code>
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

Parameter	Type	Required / Optional	Description
condition 1 — n	<code>query.Condition</code>	Required	One or more condition objects. There is no limit on the number of conditions you can specify.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});

var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
```

```

var cond3 = salesrep.createCondition({
  fieldId: 'email',
  operator: query.Operator.START_WITH_NOT,
  values: 'foo'}));

search.condition = search.and(
  cond3, search.not(
    search.or(cond1, cond2)
  )
);

var resultSet = search.run();

```

Query.autoJoin(options)

Method Description	<p>Creates a join relationship.</p> <p>Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type.</p> <p>After you create the initial query definition, use Query.autoJoin(options) to create your first join (query.Component). Then use Component.autoJoin(options) to create each subsequent join (query.Component).</p> <div> <p>Note: This method is a shortcut for the chained Query.root and Component.autoJoin(options): <code>Query.root.join(options)</code>. The Query.root property references the root component, which is a query.Component object.</p> </div> <div> <p>Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	The column type (field type) that joins the parent component to the new component. This value determines the columns on which the components are joined and the type of the newly joined component.

Parameter	Type	Required / Optional	Description
			<p>Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.autoJoin({
    fieldId: 'entity'
});

search.columns = [entity.createColumn({
    fieldId: 'subsidiary'
})];

search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
})];

var results = search.runPaged({
    pageSize: 10
});
```

Query.createColumn(options)

Method Description	<p>This method creates a query result column based on the query.Query object. The query.Column object is the equivalent of the search.Column object in the N/search Module. The query.Column object describes the field types (columns) that are displayed from the query results.</p> <p>To create columns:</p> <ul style="list-style-type: none"> ■ Use <code>Query.createColumn(options)</code> to create conditions on the initial query definition created with <code>query.create(options)</code>. Use this method in one of two ways: <ul style="list-style-type: none"> □ Pass in an argument for the parameter <code>options.fieldId</code>.
---------------------------	--

	<ul style="list-style-type: none"> □ Pass in an argument for the parameter <code>options.formula</code>. If you use this option, you can also use the optional parameter <code>options.type</code>. ■ If needed, use <code>Component.createColumn(options)</code> to create conditions on the join relationships created with <code>Query.autoJoin(options)</code> and <code>Component.autoJoin(options)</code>. ■ Assign all created columns as array values to <code>Query.columns</code>. For an example, see Syntax. <div style="border: 1px solid #0070C0; padding: 5px; margin-top: 10px;"> <p>Note: This method is a shortcut for the chained <code>Query.root</code> and <code>Component.createColumn(options)</code>: <code>Query.root.createColumn(options)</code>. The <code>Query.root</code> property references the root component, which is a <code>query.Component</code> object.</p> </div>
Returns	<code>query.Column</code> object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.fieldId</code>	string	required if <code>options.formula</code> is not used	<p>The name of the query result column. This value sets the <code>Column.fieldId</code> property.</p> <p>Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the appropriate record type. 2. Scroll until you see the Search Columns table. 3. Locate the appropriate value in the Internal ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
<code>options.formula</code>	string	required if <code>options.fieldId</code> is not used	<p>The formula used to create the query result column. This value sets the <code>Column.formula</code> property.</p> <p>For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p>
<code>options.type</code>	string	optional if <code>options.formula</code> is used	<p>If you use the <code>options.formula</code> parameter, use this parameter to explicitly define the formula's return type. Defining the formula's return type might be required if the return type cannot be determined correctly based on the specified formula. This value sets the <code>Column.type</code> property.</p> <p>Use the appropriate <code>query.ReturnType</code> enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>

Parameter	Type	Required / Optional	Description
options.aggregate	string	optional	Use this parameter to run an aggregate function on your query result column. An aggregate function performs a calculation on the column values and returns a single value. This value sets the Column.aggregate property. Use the appropriate query.Aggregate enum value to pass in your argument. This enum holds all the supported values for this parameter.
options.groupBy	boolean	optional	Indicates whether the query results are grouped by this query result column. This value sets the Column.groupBy property. If you do not pass in an argument, the default value is set to <code>false</code> .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];
```

```
var resultSet = search.run();
```

Query.createCondition(options)

Method Description	<p>This method creates a condition (query filter) based on the query.Query object. A condition narrows the query results. The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p> <ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions on the initial query definition created with query.create(options). Use this method in one of two ways: <ul style="list-style-type: none"> □ Pass in arguments for the parameters <code>options.fieldId</code>, <code>options.operator</code>, and <code>options.values</code>. The combination of these arguments translates to <i><filter column><operator><field value></i> (for example, 'city' equals 'Boston'). □ Pass in an argument for the parameter <code>options.formula</code>. If you use this option, you can also use the optional parameter <code>options.type</code>. ■ If needed, use Component.createCondition(options) to create conditions on the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ If you have multiple conditions, use them to create a new nested condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your simple or nested condition to Query.condition. For an example, see Syntax. <div>  Note: This method is a shortcut for the chained Query.root and Component.createCondition(options): <code>Query.root.createCondition(options)</code>. The Query.root property references the root component, which is a query.Component object. </div>
Returns	query.Condition object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.fieldId</code>	string	required if <code>options.operator</code> and <code>options.values</code> are used	The name of the condition. This value sets the Condition.fieldId property. Obtain this value from the Records Browser :

Parameter	Type	Required / Optional	Description
			<ol style="list-style-type: none"> 1. Go to the appropriate record type. 2. Scroll until you see the Search Filters table. 3. Locate the appropriate value in the Internal ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>
options.operator	string	required if options.fieldId and options.values are used	The operator used by the condition. This value sets the Condition.operator parameter. Use the appropriate query.Operator enum value to pass in your argument. This enum holds all the supported values for this parameter.
options.values	string[]	required if options.fieldId and options.operator are used	An array of string values. This value sets the Condition.values property.
options.formula	string	required if options.fieldId, options.operator, and options.values are not used	The formula used to create the condition. This value sets the Condition.formula property. For more information on formulas, see the help topics SuiteAnalytics Workbook Beta , SQL Expressions , and Search Formula Examples and Tips .
options.type	string	optional if options.formula is used	If you use the options.formula parameter, use this parameter to explicitly define the formula's return type. Defining the formula's return type might be required if the return type cannot be determined correctly based on the specified formula. This value sets the Condition.type property. Use the appropriate query.ReturnType enum value to pass in your argument. This enum holds all the supported values for this parameter.
options.aggregate	string	optional	Use this parameter to run an aggregate function on a condition. An aggregate function performs a calculation on the condition values and returns a single value. This value sets the Condition.aggregate property. Use the appropriate query.Aggregate enum value to pass in your argument. This enum holds all the supported values for this parameter.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

var location = salesrep.join({
    fieldId: 'location'
});
```

```

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
});

search.condition = search.and(
    cond3, search.not(
        search.or(cond1, cond2)
    )
);

var resultSet = search.run();

```

Query.createSort(options)

Method Description	<p>This method creates a sort based on the query.Query object. The query.Sort object describes a sort that is placed on a particular query result column.</p> <p>To create a sort:</p> <ul style="list-style-type: none"> Use <code>Search.createSort(options)</code> to create a sort based on the initial query definition created with query.create(options). Use <code>Component.createSort(options)</code> to create a sort based on a join relationship created with <code>Query.autoJoin(options)</code> or <code>Component.autoJoin(options)</code>. Assign all created sorts as array values to <code>Query.sort</code>. For an example, see Syntax. <div style="border: 1px solid #0070c0; padding: 10px; margin-top: 10px;"> <p> Note: This method is a shortcut for the chained <code>Query.root</code> and <code>Component.createSort(options): Query.root.createSort(options)</code>. The <code>Query.root</code> property references the root component, which is a <code>query.Component</code> object.</p> </div>
Returns	query.Sort object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.column	query.Column	required	The query result column that you want to sort by. This value sets the Sort.column property.
options.ascending	boolean	optional	Indicates whether the sort direction is ascending. This value sets the Sort.ascending property. The default value of this property is <code>true</code> , meaning that the sort direction is ascending. If you want the sort direction to be descending, set this property to <code>false</code> .
options.caseSensitive	boolean	optional	<p>Indicates whether the sort is case sensitive. This value sets the Sort.caseSensitive property. If a sort is case sensitive (and the sort direction is ascending), rows with column values that start with uppercase letters are listed before rows with column values that start with lowercase letters. If a sort is not case sensitive, uppercase and lowercase letters are treated the same. For example, the following list of items is sorted using a case-sensitive sort with a sort direction of ascending:</p> <ul style="list-style-type: none"> Banana Orange apple grapefruit kiwi <p>Here is the same list of items sorted using a regular (not case-sensitive) sort with a sort direction of ascending:</p> <ul style="list-style-type: none"> apple Banana grapefruit kiwi Orange <p>The default value of this property is <code>false</code>.</p>
options.locale	string	optional	<p>The locale to use for the sort. This value sets the Sort.locale property. A locale represents a combination of language and region, and it can affect how certain values (such as strings) are sorted. For example, languages that share the same alphabet may sort characters differently. Use this property to ensure that query results are sorted using locale-specific rules. Use the appropriate query.SortLocale enum value to pass in your argument. This enum holds all the supported values for this parameter.</p>
options.nullsLast	boolean	optional	Indicates whether query results with null values are listed at the end of the query results. This value sets the Sort.nullsLast property. The default value of this property is the value of the <code>options.ascending</code> property. For example, if the

Parameter	Type	Required / Optional	Description
			<code>options.ascending</code> property is set to <code>true</code> , the <code>options.nullsLast</code> property is also set to <code>true</code> .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();
```

Query.join(options)

Method Description	Creates a join relationship.
---------------------------	------------------------------

	<div>  Important: This method is an alias to <code>Query.autoJoin(options)</code>. Use <code>Query.autoJoin(options)</code> instead of this method to create simple joins. Use <code>Query.joinFrom(options)</code> and <code>Query.joinTo(options)</code> to create explicit directional joins. </div> <p>Use the method <code>query.create(options)</code> to create your initial query definition (<code>query.Query</code>). The initial query definition uses one search type. For available search types, see <code>query.Type</code>. After you create the initial query definition, use <code>Query.join(options)</code> to create your first join (<code>query.Component</code>). Then use <code>Component.autoJoin(options)</code> to create each subsequent join (<code>query.Component</code>).</p> <div>  Note: This method is a shortcut for the chained <code>Query.root</code> and <code>Component.join(options)</code>: <code>Query.root.join(options)</code>. The <code>Query.root</code> property references the root component, which is a <code>query.Component</code> object. </div> <div>  Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period. </div>
Returns	<code>query.Component</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	<code>query.Query</code>
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.fieldId</code>	string	required	<p>The column type (field type) that joins the parent component to the new component. This value determines the columns on which the components are joined and the type of the newly joined component. Obtain this value from the Records Browser:</p> <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. <p>For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser.</p>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.join({
    fieldId: 'entity'
});

search.columns = [entity.createColumn({
    fieldId: 'subsidiary'
})];

search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
})];

var results = search.runPaged({
    pageSize: 10
});
```

Query.joinFrom(options)

Method Description	<p>Creates an explicit directional join relationship from another component to this component (an inverse join). This method sets the Component.source property on the returned query.Component object.</p> <p>Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type. After you create the initial query definition, use this method to create your first join as an explicit directional join from another component to this component.</p> <div> <p>Note: This method is a shortcut for the chained Query.root and Component.joinFrom(options): <code>Query.root.joinFrom(options)</code>. The Query.root property references the root component, which is a query.Component object.</p> </div> <div> <p>Important: For the 2018.2 beta release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query

Sibling Object Members	Query Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	The column type (field type) that joins the parent component to the new component. Obtain this value from the Records Browser : <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser .
options.source	string	required	The search type of the component joined to this component. This value sets the Component.source property. This value can be described as the inverse relationship of this component, and it determines the source search type of the newly joined component.

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Query.joinTo(options)

Method Description	<p>Creates an explicit directional join relationship to another component from this component (a polymorphic join). This method sets the Component.target property on the returned query.Component object.</p> <p>Use the method query.create(options) to create your initial query definition (query.Query). The initial query definition uses one search type. For available search types, see query.Type.</p> <p>After you create the initial query definition, use this method to create your first join as an explicit directional join to another component from this component.</p> <div data-bbox="410 1564 1365 1677"> <p> Note: This method is a shortcut for the chained Query.root and Component.joinTo(options): <code>Query.root.autoJoin(options)</code>. The Query.root property references the root component, which is a query.Component object.</p> </div> <div data-bbox="410 1696 1365 1841"> <p> Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period.</p> </div>
Returns	query.Component object

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.fieldId	string	required	The column type (field type) that joins the parent component to the new component. Obtain this value from the Records Browser : <ol style="list-style-type: none"> 1. Go to the parent component's record type. 2. Scroll until you see the Search Joins table. 3. Locate the appropriate value in the Join ID column. For more information on the Records Browser, see the help topic Using the SuiteScript Records Browser .
options.target	string	required	The search type of the component joined to this component. This value sets the Component.target property. This value can be described as the polymorphic relationship of this component, and it determines the target search type of the newly joined component.

Errors

Error Code	Thrown If
RELATIONSHIP_ALREADY_USED	The specified join relationship already exists.

Query.run()

Method Description	Executes the query and returns the query result set.
Returns	query.ResultSet
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 Usage Units
Module	N/query Module
Parent Object	query.Query

Sibling Object Members	Query Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();
```

Query.run.promise()

Method Description	Executes the query asynchronously and returns the query result set.
Returns	query.ResultSet

Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 Usage Units
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Query.runPaged()

Method Description	Executes the query and returns a set of paged results.
Returns	query.PagedData
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 Usage Units
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.TRANSACTION
});

var entity = search.join({
    fieldId: 'entity'
});

search.columns = [entity.createColumn({
    name: 'subsidiary'
})];

search.sort = [search.createSort({
    column: search.columns[0],
    ascending: false
})];

var results = search.runPaged({
    pageSize: 10
});
```

```
// Use the count property to count the
// search results easily
var resultCount = search.runPaged({
    pageSize: 10
}).count;
```

Query.runPaged.promise()

Method Description	Executes the query asynchronously and returns a set of paged results.
Returns	query.PagedData
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 Usage Units
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Query.not()

Method Description	<p>Creates a new condition (a query.Condition object) that corresponds to a logical negation (NOT) of the argument passed to the method. The argument must be a query.Condition object. A condition narrows the query results. The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p> <ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions for the initial query definition created with query.create(options). ■ Use Component.createCondition(options) to create conditions for the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ If you have multiple conditions, use them to create a new parent condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your parent condition to Query.condition. For an example, see Syntax.
Returns	query.Condition
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

Parameter	Type	Required / Optional	Description
condition	query.Condition	Required	One condition object.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});
var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo' });

search.condition = search.and(
    cond3, search.not(
        search.or(cond1, cond2)
    )
);

var resultSet = search.run();
```

Query.or()

Method Description	<p>Creates a new condition (a query.Condition object) that corresponds to a logical disjunction (OR) of the arguments passed to the method. The arguments must be one or more query.Condition objects.</p> <p>A condition narrows the query results. The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p>
---------------------------	--

	<ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions for the initial query definition created with query.create(options). ■ Use Component.createCondition(options) to create conditions for the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ If you have multiple conditions, use them to create a new parent condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your parent condition to Query.condition. For an example, see Syntax.
Returns	query.Condition object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Parameters

Parameter	Type	Required / Optional	Description
condition 1 — n	query.Condition	Required	One or more condition objects. There is no limit on the number of conditions you can specify.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});
var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
```



```

        operator: query.Operator.EQUAL,
        values: 2647
    });
    var cond3 = salesrep.createCondition({
        fieldId: 'email',
        operator: query.Operator.START_WITH_NOT,
        values: 'foo' });

    search.condition = search.and(
        cond3, search.not(
            search.or(cond1, cond2)
        )
    );

    var resultSet = search.run();

```

Query.child

Property Description	Holds a references to children of this component. The value of this property is an object of key/value pairs. Each key is the name of a child component. Each respective value is the corresponding query.Component object. The object values are set with the execution of Query.autoJoin(options) and Component.autoJoin(options) . The order of the key/value pairs reflects the parent/child hierarchy.
Type	Object
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Query.columns

Property Description	<p>Holds an array of result columns (query.Column objects) returned from the query. The query.Column object is the equivalent of the search.Column object in the N/search Module. The query.Column object describes a field type (column) that is returned from the query results. To create columns:</p> <ul style="list-style-type: none"> ■ Use Query.createColumn(options) to create conditions on the initial query definition created with query.create(options). ■ Use Component.createColumn(options) to create conditions on the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ Assign all created columns as array values to <code>Query.columns</code>. For an example, see Syntax.
Type	query.Column[]
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members

Since	2018.1
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
        column: search.columns[1]
    }),
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();
```

Query.condition

Property Description	<p>References the simple or nested condition (a query.Condition object) that narrows the query results.</p> <p>The query.Condition object acts in the same capacity as the search.Filter object in the N/search Module. The primary difference is that query.Condition objects can contain other query.Condition objects.</p> <p>To create conditions:</p>
-----------------------------	--

	<ul style="list-style-type: none"> ■ Use Query.createCondition(options) to create conditions for the initial query definition created with query.create(options). ■ Use Component.createCondition(options) to create conditions for the join relationships created with Query.autoJoin(options) and Component.autoJoin(options). ■ If you have multiple conditions, use them to create a new nested condition with the methods Query.and(), Query.or(), and Query.not(). ■ Assign your simple or nested condition to <code>Query.condition</code>. For an example, see Syntax.
Type	query.Condition object
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});
var location = salesrep.join({
    fieldId: 'location'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
});

search.condition = search.and(
    cond3, search.not(
        search.or(cond1, cond2)
    )
)
```

```
);  
  
var resultSet = search.run();
```

Query.id

Property Description	<p>Holds the ID of the query definition.</p> <p>This property has a value only for existing queries that are loaded using query.load(options). If you create a query using query.create(options) but do not save it, this property is null.</p> <div>  Important: In the 2018.2 release, you can use the N/query module to load and delete existing searches, but you cannot save searches. You can save searches using the SuiteAnalytics Workbook UI. </div>
Type	number (read-only)
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Query.name

Property Description	<p>Holds the name of the query definition.</p> <p>This property has a value only for existing queries that are loaded using query.load(options). If you create a query using query.create(options) but do not save it, this property is null.</p> <div>  Important: In the 2018.2 release, you can use the N/query module to load and delete existing searches, but you cannot save searches. You can save searches using the SuiteAnalytics Workbook UI. </div>
Type	string (read-only)
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Query.root

Property Description	<p>References the root component of the query definition.</p> <p>The initial query.Component object is called the root component. It encapsulates the initial search type passed to query.create(options). The root component is automatically created with the query.Query object and is a child of the query.Query object.</p>
Type	query.Component (read-only)
Module	N/query Module
Parent Object	query.Query

Sibling Object Members	Query Object Members
Since	2018.1

Query.sort

Property Description	<p>Holds an array of query result columns (query.Column objects) used for sorting. This object encapsulates a sort based on the query.Query or query.Component object. The query.Sort object describes a sort that is placed on a particular query result column. To create a sort:</p> <ul style="list-style-type: none"> ■ Use Query.createSort(options) to create a sort based on the initial query definition created with query.create(options). ■ Use Component.createSort(options) to create a sort based on a join relationship created with Query.autoJoin(options) or Component.autoJoin(options). ■ Assign all created sorts as array values to Query.sort. For an example, see Syntax.
Type	query.Sort[]
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    })
];
```

```

    }},
  ];

  search.sort = [
    search.createSort({
      column: search.columns[1]
    }),
    salesrep.createSort({
      column: salesrep.columns[0],
      ascending: false
    })
  ];

  var resultSet = search.run();

```

Query.type

Property Description	Describes the initial search type of the query definition. This property is set during the execution of query.create(options) .
Type	string (read-only)
Module	N/query Module
Parent Object	query.Query
Sibling Object Members	Query Object Members
Since	2018.1

query.Result

Object Description	Encapsulates a single row of the result set (query.ResultSet).
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Result Object Members
Since	2018.1

Result.columns

Property Description	Holds an array of query return column references. These array values are equivalent to the array values in ResultSet.columns .
Type	query.Column[] (read-only)
Module	N/query Module
Parent Object	query.Result
Sibling Object Members	Result Object Members
Since	2018.1

Result.values

Property Description	Describes the result values. Value types correspond to the ResultSet.types property. Array values correspond to the array values for ResultSet.columns and Result.columns .
Type	string[] or number[] or boolean[] (read-only)
Module	N/query Module
Parent Object	query.Result
Sibling Object Members	Result Object Members
Since	2018.1

query.ResultSet

Object Description	Encapsulates the set of results returned by the query. Use Query.run() or Query.run.promise() to create this object.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	ResultSet Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var resultSet = search.run();
var results = resultSet.results;
for (var i = results.length - 1; i >= 0; i--)
    log.debug(results[i].values);
```

ResultSet.iterator()

Method Description	Standard SuiteScript 2.0 object for iterating through results
Returns	Iterator object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Parent Object	query.ResultSet
Sibling Object Members	ResultSet Object Members

Since	2018.1
-------	--------

ResultSet.columns

Property Description	Holds an array of query return column references. The <code>ResultSet.columns</code> array values correspond with the ResultSet.types array values.
Type	query.Column[] (read-only)
Module	N/query Module
Parent Object	query.ResultSet
Sibling Object Members	ResultSet Object Members
Since	2018.1

ResultSet.results

Property Description	Holds an array of query.Result objects.
Type	query.Result[] (read-only)
Module	N/query Module
Parent Object	query.ResultSet
Sibling Object Members	ResultSet Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var resultSet = search.run();
var results = resultSet.results;
for (var i = results.length - 1; i >= 0; i--)
    log.debug(results[i].values);
```

ResultSet.types

Property Description	Holds an array of the return types for ResultSet.results . The <code>ResultSet.types</code> array values correspond with the ResultSet.columns array values.
Type	string[] (read-only)
Module	N/query Module
Parent Object	query.ResultSet
Sibling Object Members	ResultSet Object Members
Since	2018.1

query.Sort

Object Description	<p>Encapsulates a sort based on the query.Query or query.Component object. The <code>query.Sort</code> object describes a sort that is placed on a particular query result column. To create a sort:</p> <ul style="list-style-type: none"> ■ Use Query.createSort(options) to create a sort based on the initial query definition created with query.create(options). ■ Use Component.createSort(options) to create a sort based on a join relationship created with Query.autoJoin(options) or Component.autoJoin(options). ■ Assign all created sorts as array values to Query.sort. For an example, see Syntax.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/query Module
Methods and Properties	Sort Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    }),
    search.createColumn({
        fieldId: 'id'
    }),
    salesrep.createColumn({
        fieldId: 'entityid'
    }),
    salesrep.createColumn({
        fieldId: 'email'
    }),
    salesrep.createColumn({
        fieldId: 'hiredate'
    }),
];

search.sort = [
    search.createSort({
```

```

        column: search.columns[1]
    }},
    salesrep.createSort({
        column: salesrep.columns[0],
        ascending: false
    })
];

var resultSet = search.run();

```

Sort.ascending

Property Description	Indicates whether the sort direction is ascending. This property is set during the execution of Query.createSort(options) and Component.createSort(options) . The default value of this property is <code>true</code> , meaning that the sort direction is ascending. If you want the sort direction to be descending, set this property to <code>false</code> .
Type	boolean
Module	N/query Module
Parent Object	query.Sort
Sibling Object Members	Sort Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```

var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false,
        caseSensitive: true,
        locale: query.SortLocale.EN_CA,
        nullsLast: false
    })
];

var resultSet = search.run();

```

Sort.caseSensitive

Property Description	<p>Indicates whether the sort is case sensitive.</p> <p>This property is set during the execution of Query.createSort(options) and Component.createSort(options).</p> <p>If a sort is case sensitive (and the sort direction is ascending), rows with column values that start with uppercase letters are listed before rows with column values that start with lowercase letters. If a sort is not case sensitive, uppercase and lowercase letters are treated the same. For example, the following list of items is sorted using a case-sensitive sort with a sort direction of ascending:</p> <ul style="list-style-type: none"> ■ Banana ■ Orange ■ apple ■ grapefruit ■ kiwi <p>Here is the same list of items sorted using a regular (not case-sensitive) sort with a sort direction of ascending:</p> <ul style="list-style-type: none"> ■ apple ■ Banana ■ grapefruit ■ kiwi ■ Orange <p>The default value of this property is <code>false</code>.</p>
Type	boolean
Module	N/query Module
Parent Object	query.Sort
Sibling Object Members	Sort Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
```

```

search.createSort({
    column: search.columns[0],
    ascending: false,
    caseSensitive: true,
    locale: query.SortLocale.EN_CA,
    nullsLast: false
})
};

var resultSet = search.run();

```

Sort.column

Property Description	Describes the query result column that the query results are sorted by. This property is set during the execution of Query.createSort(options) and Component.createSort(options) .
Type	query.Column (read-only)
Module	N/query Module
Parent Object	query.Sort
Sibling Object Members	Sort Object Members
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```

var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false,
        caseSensitive: true,
        locale: query.SortLocale.EN_CA,
        nullsLast: false
    })
];

var resultSet = search.run();

```

Sort.locale

Property Description	The locale to use for the sort. This property uses values from the query.SortLocale enum. This property is set during the execution of Query.createSort(options) and Component.createSort(options) . A locale represents a combination of language and region, and it can affect how certain values (such as strings) are sorted. For example, languages that share the same alphabet may sort characters differently. Use this property to ensure that query results are sorted using locale-specific rules.
Type	string
Module	N/query Module
Parent Object	query.Sort
Sibling Object Members	Sort Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false,
        caseSensitive: true,
        locale: query.SortLocale.EN_CA,
        nullsLast: false
    })
];

var resultSet = search.run();
```

Sort.nullsLast

Property Description	Indicates whether query results with null values are listed at the end of the query results. This property is set during the execution of Query.createSort(options) and Component.createSort(options) . The default value of this property is the value of the Sort.ascending property. For example, if the Sort.ascending property is set to <code>true</code> , the <code>Sort.nullsLast</code> property is also set to <code>true</code> .
Type	boolean

Module	N/query Module
Parent Object	query.Sort
Sibling Object Members	Sort Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false,
        caseSensitive: true,
        locale: query.SortLocale.EN_CA,
        nullsLast: false
    })
];

var resultSet = search.run();
```

query.create(options)

Method Description	Creates a query.Query object. Use this method to create your initial query definition. The initial query definition uses one search type. For available search types, see query.Type . After you create the initial query definition, use Query.autoJoin(options) to create your first join. Then use Component.autoJoin(options) to create all subsequent joins.
Returns	query.Query object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.type	string	required	<p>The search type that you want to use for the initial query definition. Use the query.Type enum to set this value (for an example, see Syntax). When you execute <code>query.create(options)</code>, the Query.type property is set based on this value.</p> <div>  Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics SuiteAnalytics Workbook Beta and Supported Record Types for the SuiteAnalytics Workbook Beta Period. </div>

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
  type: query.Type.CUSTOMER
});

var salesrep = search.join({
  fieldId: 'salesrep'
});

search.columns = [
  search.createColumn({
    fieldId: 'entityid'
  }),
  search.createColumn({
    fieldId: 'id'
  }),
  salesrep.createColumn({
    fieldId: 'entityid'
  }),
  salesrep.createColumn({
    fieldId: 'email'
  }),
  salesrep.createColumn({
    fieldId: 'hiredate'
  }),
];

search.sort = [
  search.createSort({
    column: search.columns[1]
  }),
  salesrep.createSort({
```

```

        column: salesrep.columns[0],
        ascending: false
    })
};

var resultSet = search.run();

```

query.delete(options)

Method Description	<p>Deletes an existing query. Use this method to delete a query definition that was previously created using the SuiteAnalytics Workbook UI. After the query is deleted, it is no longer available and cannot be modified or executed.</p> <div>  Important: In the 2018.2 release, you can use the N/query module to load and delete existing searches, but you cannot save searches. You can save searches using the SuiteAnalytics Workbook UI. </div>
Returns	void
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	5 Usage Units
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	number	required	The ID of the query to delete.

Errors

Error Code	Thrown If
UNABLE_TO_DELETE_QUERY	A query with the specified ID cannot be deleted because the query does not exist or you do not have permission to delete it.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```

var deletedSearch = query.delete({
    id: 237
});

```



```
});
```

query.load(options)

Method Description	<p>Loads an existing query as a query.Query object. Use this method to load a query definition that was previously created using the SuiteAnalytics Workbook UI. After the query is loaded, you can modify the query definition (for example, by setting additional property values), join the query definition with other search types, and execute the query in the same way as queries that you create using query.create(options).</p> <div>  Important: In the 2018.2 release, you can use the N/query module to load and delete existing searches, but you cannot save searches. You can save searches using the SuiteAnalytics Workbook UI. </div>
Returns	query.Query object
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	5 Usage Units
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	number	required	The ID of the query to load.

Errors

Error Code	Thrown If
UNABLE_TO_LOAD_QUERY	A query with the specified ID cannot be loaded because the query does not exist or you do not have permission to load it.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.load({
  id: 237
});

var salesrep = search.autoJoin({
  fieldId: 'salesrep'
```

```
});

var resultSet = search.run();
```

query.Aggregate

Enum Description	<p>Holds the string values for aggregate functions supported with the N/query Module. An aggregate function performs a calculation on the column or condition values and returns a single value.</p> <p>Each value in this enum (except <code>MEDIAN</code>) has two variants: distinct (using the <code>_DISTINCT</code> suffix) and nondistinct (using no suffix). The variant determines whether the aggregate function operates on all instances of duplicate values or on just a single instance of the value. For example, consider a situation in which the <code>MAXIMUM</code> aggregate function is used to determine the maximum of a set of values. When using the distinct variant (<code>MAXIMUM_DISTINCT</code>), the aggregate function considers each instance of duplicate values. So if the set of values includes three distinct values that are all equal and all represent the maximum value in the set, the aggregate function lists all three instances. When using the nondistinct variant (<code>MAXIMUM</code>), only one instance of the maximum value is listed, regardless of the number of instances of that maximum value in the set.</p> <p>This enum is used to pass the aggregate function argument to <code>Component.createColumn(options)</code>, <code>Component.createCondition(options)</code>, <code>Query.createColumn(options)</code>, and <code>Query.createCondition(options)</code>.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.1

Values

Value	Description
AVERAGE	Calculates the average value.
AVERAGE_DISTINCT	Calculates the average distinct value.
COUNT	Counts the number of results.
COUNT_DISTINCT	Counts the number of distinct results.
MAXIMUM	Determines the maximum value. If the values are dates, the most recent date is determined.
MAXIMUM_DISTINCT	Determines the maximum distinct value. If the values are dates, the most recent date is determined.
MEDIAN	Calculates the median value.
MINIMUM	Determines the minimum value. If the values are dates, the earliest date is determined.

Value	Description
MINIMUM_DISTINCT	Determines the minimum distinct value. If the values are dates, the earliest date is determined.
SUM	Adds all values.
SUM_DISTINCT	Adds all distinct values.

query.Operator

Enum Description	<p>Holds the string values for operators supported with the N/query Module. This enum is used to pass the operator argument to Query.createCondition(options) and Component.createCondition(options).</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.1

Values

Value
AFTER
AFTER_NOT
ANY_OF
ANY_OF_NOT
BEFORE
BEFORE_NOT
BETWEEN
BETWEEN_NOT
CONTAIN
CONTAIN_NOT
EMPTY
EMPTY_NOT
ENDWITH
ENDWITH_NOT
EQUAL

Value
EQUAL_NOT
GREATER
GREATER_NOT
GREATER_OR_EQUAL
GREATER_OR_EQUAL_NOT
IS
IS_NOT
LESS
LESS_NOT
LESS_OR_EQUAL
LESS_OR_EQUAL_NOT
ON
ON_NOT
ON_OR_AFTER
ON_OR_AFTER_NOT
ON_OR_BEFORE
ON_OR_BEFORE_NOT
START_WITH
START_WITH_NOT
WITHIN
WITHIN_NOT

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.join({
    fieldId: 'salesrep'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
```

```

    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
  });

  var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
  });

  search.condition = search.and(
    cond3, search.not(
      search.or(cond1, cond2)
    )
  );

  var resultSet = search.run();

```

query.ReturnType

Enum Description	<p>Holds the string values for the formula return types supported with the N/query Module. This enum is used to pass the formula return type argument to Query.createColumn(options), Component.createColumn(options), Query.createCondition(options), and Component.createCondition(options). For more information on formulas, see the help topics SuiteAnalytics Workbook Beta, SQL Expressions, and Search Formula Examples and Tips.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.1

Values

Value
ANY
BOOLEAN
CURRENCY
DATE
DATETIME
DURATION

Value
FLOAT
INTEGER
KEY
RELATIONSHIP
STRING
UNKNOWN

query.SortLocale

Enum Description	<p>Holds the string values for sort locales supported with the N/query Module. This enum is used to pass the locale argument to Query.createSort(options) and Component.createSort(options).</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.2

Values

Value
ARABIC
ARABIC_ABJ_MATCH
ARABIC_ABJ_MATCH_CI
ARABIC_ABJ_SORT
ARABIC_ABJ_SORT_CI
ARABIC_CI
ARABIC_MATCH
ARABIC_MATCH_CI
ASCII7
ASCII7_CI
AZERBAIJANI
AZERBAIJANI_CI
BENGALI

Value
BENGALI_CI
BIG5
BIG5_CI
BINARY
BINARY_CI
BULGARIAN
BULGARIAN_CI
CANADIAN_M
CATALAN
CATALAN_CI
CROATIAN
CROATIAN_CI
CS_CZ
CZECH
CZECH_CI
CZECH_PUNCTUATION
CZECH_PUNCTUATION_CI
DANISH
DANISH_CI
DANISH_M
DA_DK
DE_DE
DUTCH
DUTCH_CI
EBCDIC
EBCDIC_CI
EEC_EURO
EEC_EUROPA3
EEC_EUROPA3_CI
EEC_EURO_CI
EN
EN_AU
EN_CA
EN_GB

Value
EN_US
ESTONIAN
ESTONIAN_CI
ES_AR
ES_ES
FINNISH
FINNISH_CI
FRENCH
FRENCH_AI
FRENCH_CI
FRENCH_M
FR_CA
FR_FR
GBK
GBK_AI
GBK_CI
GENERIC_M
GERMAN
GERMAN_AI
GERMAN_CI
GERMAN_DIN
GERMAN_DIN_AI
GERMAN_DIN_CI
GREEK
GREEK_AI
GREEK_CI
HEBREW
HEBREW_AI
HEBREW_CI
HE_IL
HKSCS
HKSCS_AI
HKSCS_CI
HUNGARIAN

Value
HUNGARIAN_AI
HUNGARIAN_CI
ICELANDIC
ICELANDIC_AI
ICELANDIC_CI
INDONESIAN
INDONESIAN_AI
INDONESIAN_CI
ITALIAN
ITALIAN_AI
ITALIAN_CI
IT_IT
JAPANESE_M
JA_JP
KOREAN_M
KO_KR
LATIN
LATIN_AI
LATIN_CI
LATVIAN
LATVIAN_AI
LATVIAN_CI
LITHUANIAN
LITHUANIAN_AI
LITHUANIAN_CI
MALAY
MALAY_AI
MALAY_CI
NL_NL
NORWEGIAN
NORWEGIAN_AI
NORWEGIAN_CI
POLISH
POLISH_AI

Value
POLISH_CI
PT_BR
PUNCTUATION
PUNCTUATION_AI
PUNCTUATION_CI
ROMANIAN
ROMANIAN_AI
ROMANIAN_CI
RUSSIAN
RUSSIAN_AI
RUSSIAN_CI
RU_RU
SCHINESE_PINYIN_M
SCHINESE_RADICAL_M
SCHINESE_STROKE_M
SLOVAK
SLOVAK_AI
SLOVAK_CI
SLOVENIAN
SLOVENIAN_AI
SLOVENIAN_CI
SPANISH
SPANISH_AI
SPANISH_CI
SPANISH_M
SV_SE
SWEDISH
SWEDISH_AI
SWEDISH_CI
SWISS
SWISS_AI
SWISS_CI
TCHINESE_RADICAL_M
TCHINESE_STROKE_M

Value
THAI_M
TH_TH
TR_TR
TURKISH
TURKISH_AI
TURKISH_CI
UKRAINIAN
UKRAINIAN_AI
UKRAINIAN_CI
UNICODE_BINARY
UNICODE_BINARY_AI
UNICODE_BINARY_CI
VIETNAMESE
VIETNAMESE_AI
VIETNAMESE_CI
WEST_EUROPEAN
WEST_EUROPEAN_AI
WEST_EUROPEAN_CI
ZH_CN
ZH_TW

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

search.columns = [
    search.createColumn({
        fieldId: 'entityid'
    })
];

search.sort = [
    search.createSort({
        column: search.columns[0],
        ascending: false,
        caseSensitive: true,
```

```

        locale: query.SortLocale.EN_CA,
        nullsLast: false
    })
};

var resultSet = search.run();

```

query.Type

Important: For the 2018.2 release, the N/query module supports the same record types supported by the SuiteAnalytics Workbook UI. For more information, see the help topics [SuiteAnalytics Workbook Beta](#) and [Supported Record Types for the SuiteAnalytics Workbook Beta Period](#).

Enum Description	<p>Holds the string values for search types used in the query definition. This enum is used to pass the initial search type argument to query.create(options).</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Type	enum
Module	N/query Module
Sibling Module Members	N/query Module Members
Since	2018.1

Values

Note: Before using these values, consider the following:

- A search type is not the same as a record type. The supported search types listed below do not necessarily correspond with the supported record types listed in the [N/record Module](#).
- Depending on your account and role, some of these values might not be available.

Enum Value	Sets Query.type Property To
ACCOUNT	account
ACCOUNTING_CONTEXT	accountingcontext
ACCOUNTING_PERIOD	accountingperiod
ADVANCED_REV_REC_PLUGIN	advancedrevrecplugin
ADV_INTERCOMPANY_JOURNAL_ENTRY	advintercompanyjournalentry
ALLOCATION_METHOD	allocationmethod
AMORTIZATION_SCHEDULE	amortizationschedule
AMORTIZATION_TEMPLATE	amortizationtemplate
ANOTHER_HIERARCHY_RECORD	anotherhierarchyrecord

Enum Value	Sets Query.type Property To
BANK_CONNECTIVITY_PLUGIN	bankconnectivityplugin
BILLING_CLASS	billingclass
BILLING_SCHEDULE	billingschedule
BRANCHRECORD	branchrecord
BUDGETCATEGORY	budgetcategory
BUDGETEXCHANGERATE	budgetexchangerate
BUDGETIMPORT	budgetimport
BUDGETS	budgets
BULK_PROC_SUBMISSION	bulkprocsubmission
BUNDLE_INSTALLATION_SCRIPT	bundleinstallationscript
BUNDLE_INSTALLATION_SCRIPT_DEPLOYMENT	bundleinstallationscriptdeployment
BUYING_REASON	buyingreason
BUYING_TIME_FRAME	buyingtimeframe
CALENDAR_EVENT	calendarevent
CAMPAIGN_AUDIENCE	campaignaudience
CAMPAIGN_CATEGORY	campaigncategory
CAMPAIGN_CHANNEL	campaignchannel
CAMPAIGN_EMAIL_ADDRESS	campaignemailaddress
CAMPAIGN_EVENT	campaignevent
CAMPAIGN_FAMILY	campaignfamily
CAMPAIGN_OFFER	campaignoffer
CAMPAIGN_RESPONSE	campaignresponse
CAMPAIGN_SEARCH_ENGINE	campaignsearchengine
CAMPAIGN_TEMPLATE	campaigntemplate
CAMPAIGN_VERTICAL	campaignvertical
CASE_PROFILE	caseprofile
CASH_REFUND	cashrefund
CASH_SALE	cashsale
CATEGORY1099MISC	category1099misc
CHECK	check
CLASSIFICATION	classification
CLIENT_SCRIPT	clientscript
CLIENT_SCRIPT_DEPLOYMENT	clientscriptdeployment
CLOB_RECORD	clobrecord

Enum Value	Sets Query.type Property To
COMPANY	company
COMPETITOR	competitor
COMPOSITE_KEY_SOURCE_RECORD	compositekeysource record
COMPOSITE_RECORD	compositerecord
CONSOLIDATEDEXCHANGERATE	consolidatedexchangerate
CONSOLIDATEDEXCHANGERATEINTERNAL	consolidatedexchangerateinternal
CONSOLIDATED_RATE_ADJUSTOR_PLUGIN	consolidatedrateadjustorplugin
CONSOLIDATION_ACCOUNT	consolidationaccount
CONSOLIDATION_ACCOUNT_TYPE	consolidationaccounttype
CONSOLIDATION_BUDGET_RATE	consolidationbudgetrate
CONSOLIDATION_CURRENCY	consolidationcurrency
CONSOLIDATION_RATE	consolidationrate
CONSOLIDATION_SUBSIDIARY	consolidationsubsidiary
CONSOLIDATION_TRANSACTION	consolidationtransaction
CONSUMER_SPECIFIC_RECORD_TYPE	consumerspecificrecordtype
CONTACT	contact
CONTACT_CATEGORY	contactcategory
CONTACT_ROLE	contactrole
COUPON_CODE	couponcode
COURSE_RECORD	courserecord
CREDIT_CARDS	creditcards
CREDIT_CARD_CHARGE	creditcardcharge
CREDIT_CARD_REFUND	creditcardrefund
CREDIT_MEMO	creditmemo
CRM_TEMPLATE	crmtemplate
CRM_TEMPLATE_CATEGORY	crmtemplatecategory
CURRENCY	currency
CURRENCY_FIELD_RECORD	currencyfieldrecord
CURRENCY_FIELD_TYPE	currencyfieldtype
CURRENCY_RATE	currencyrate
CUSTOM	custom
CUSTOMER	customer
CUSTOMER_CATEGORY	customercategory
CUSTOMER_CHARGE	customercharge

Enum Value	Sets Query.type Property To
CUSTOMER_DEPOSIT	customerdeposit
CUSTOMER_MESSAGE	customermessage
CUSTOMER_PAYMENT	customerpayment
CUSTOMER_REFUND	customerrefund
CUSTOMER_STATUS	customerstatus
CUSTOMRECORD1	customrecord1
CUSTOM_GL_PLUGIN	customglplugin
CUSTOM_LIST	customlist
CUSTOM_RECORD_TYPE	customrecordtype
DATE_FIELD_TYPE	datefieldtype
DATE_RECORD	daterecord
DATE_TIME_RECORD	datetimerecord
DATE_TIME_ZONE	datetimezone
DEFAULTING_PORTED_RECORD	defaultingportedrecord
DEF_VIEW_TEST_RECORD	defviewtestrecord
DELETED_RECORD	deletedrecord
DEPARTMENT	department
DEPOSIT	deposit
DEPOSIT_APPLICATION	depositapplication
DESCRIPTION_ITEM	descriptionitem
DEVICE_ID	deviceid
DISABLEDCHANNELFORMTESTRECORD	disabledchannelformtestrecord
DISCOUNT_ITEM	discountitem
DISPLAY_INACTIVE_TEST_RECORD	displayinactivetestrecord
DOMAIN	domain
DOWNLOAD_ITEM	downloaditem
DURATION_RECORD	durationrecord
EMAIL_CAPTURE_PLUGIN	emailcaptureplugin
EMAIL_TEMPLATE	emailtemplate
EMPLOYEE	employee
EMPLOYEE_LIST	employeeelist
EMPLOYEE_STATUS	employeeestatus
END_TO_END_TIME	endtoendtime
ENTITY	entity

Enum Value	Sets Query.type Property To
ENTITY_GROUP	entitygroup
ESCALATION_TERRITORY	escalationterritory
ESTIMATE	estimate
EXAMPLE_TRANSACTION	exampletransaction
EXPENSE_CATEGORY	expensecategory
EXPENSE_REPORT	expensereport
EXPOSURENOTLIMITEDRECORD	exposurenolimitedrecord
FACULTYRECORD	facultyrecord
FAX_TEMPLATE	faxtemplate
FIELD_LABEL	fieldlabel
FILE	file
FLOAT_NUMBERS_TEST_RECORD	floatnumberstestrecord
FORECAST	forecast
FORMULA_POLYMORPHIC_RECORD	formulapolymorphicrecord
FORMULA_RECORD	formularecord
FULFILLMENT_EXCEPTION_REASON	fulfillmentexceptionreason
FX_REVAL	fxreval
GATEWAY_NOTIFICATION	gatewaynotification
GENERAL_ALLOCATION_SCHEDULE	generalallocationschedule
GENERIC_RESOURCE	genericresource
GENERIC_TEST_RECORD	generictestrecord
GIFT_CERTIFICATE	giftcertificate
GIFT_CERTIFICATE_ITEM	giftcertificateitem
HIERARCHY_RECORD	hierarchyrecord
HYBRID_RECORD_LOG	hybridrecordlog
INCOTERM	incoterm
INTEGRATION_APP	integrationapp
INTERNAL_ID_TEST_RECORD	internalidtestrecord
INVENTORY_ADJUSTMENT	inventoryadjustment
INVENTORY_DISTRIBUTION	inventorydistribution
INVENTORY_ITEM	inventoryitem
INVENTORY_TRANSFER	inventorytransfer
INVENTORY_WORKSHEET	inventoryworksheet
INVOICE	invoice

Enum Value	Sets Query.type Property To
INVT_ITEM_PRICE_HISTORY	invtitempricehistory
ISSUE	issue
ISSUE_EXTERNAL_STATUS	issueexternalstatus
ISSUE_PRIORITY	issuepriority
ISSUE_PRODUCT	issueproduct
ISSUE_REPRODUCIBILITY	issuereproducibility
ISSUE_ROLE	issuerole
ISSUE_SEVERITY	issueseverity
ISSUE_SOURCE	issuesource
ISSUE_STATUS	issuestatus
ISSUE_TAG	issuetag
ISSUE_TRACK_CODE	issuetrackcode
ISSUE_TYPE	issuetype
ITEM	item
ITEM_FULFILLMENT	itemfulfillment
ITEM_GROUP	itemgroup
ITEM_RECEIPT	itemreceipt
I_P_RESTRICTIONS	iprestrictions
JOB	job
JOB_RESOURCE_ROLE	jobresourcerole
JOB_STATUS	jobstatus
JOB_TYPE	jobtype
JOURNAL	journal
KIT_ITEM	kititem
KNOWLEDGE_BASE	knowledgebase
LOCATION	location
LOCATION_COSTING_GROUP	locationcostinggroup
LOGIN_AUDIT	loginaudit
MAIL_TEMPLATE	mailtemplate
MAP_REDUCE_SCRIPT	mapreducescript
MAP_REDUCE_SCRIPT_DEPLOYMENT	mapreducescriptdeployment
MARKUP_ITEM	markupitem
MASS_UPDATE_SCRIPT	massupdatescript
MASS_UPDATE_SCRIPT_DEPLOYMENT	massupdatescriptdeployment

Enum Value	Sets Query.type Property To
MATERIALIZED_HIERARCHY_RECORD	materializedhierarchyrecord
MEDIA_ITEM_FOLDER	mediaitemfolder
MEM_DOC	memdoc
MEM_DOC_TRANSACTION_TEMPLATE	memdoctransactiontemplate
MESSAGE	message
NAMED_GROUP_RECORD	namedgrouprecord
NEXUS	nexus
NON_INVENTORY_PURCHASE_ITEM	noninventorypurchaseitem
NON_INVENTORY_RESALE_ITEM	noninventoryresaleitem
NON_INVENTORY_SALE_ITEM	noninventorysaleitem
NOTE	note
NOTE_TYPE	notetype
NUMERIC_RECORD	numericrecord
ONLINE_CASE_FORM	onlinecaseform
ONLINE_FORM_TEMPLATE	onlineformtemplate
ONLINE_LEAD_FORM	onlineleadform
OPPORTUNITY	opportunity
OTHER_CHARGE_PURCHASE_ITEM	otherchargepurchaseitem
OTHER_CHARGE_RESALE_ITEM	otherchargeresaleitem
OTHER_CHARGE_SALE_ITEM	otherchargesaleitem
OTHER_NAME	othername
OTHER_NAME_CATEGORY	othernamecategory
PAGE	page
PAGINATION_RECORD	paginationrecord
PARTNER	partner
PARTNER_CATEGORY	partnercategory
PAYCHECK	paycheck
PAYMENT_EVENT	paymentevent
PAYMENT_GATEWAY_PLUGIN	paymentgatewayplugin
PAYMENT_ITEM	paymentitem
PAYMENT_METHOD	paymentmethod
PAYMENT_PROCESSING_PROFILE	paymentprocessingprofile
PAYROLL_ITEM	payrollitem
PDF_TEMPLATE	pdftemplate

Enum Value	Sets Query.type Property To
PERSISTED_RECORD	persistedrecord
PERSISTED_RECORD_FULL_JOIN	persistedrecordfulljoin
PERSISTED_RECORD_INVALID_TABLE	persistedrecordinvalidtable
PERSISTED_RECORD_NO_CREATE	persistedrecordnocreate
PERSISTED_RECORD_NO_DELETE	persistedrecordnodelete
PERSISTED_RECORD_NO_EDIT	persistedrecordnoedit
PERSISTED_RECORD_NO_LOAD	persistedrecordnoload
PERSISTED_RECORD_RIGHT_JOIN	persistedrecordrightjoin
PERSISTED_RECORD_SIMPLE_OPTIONS	persistedrecordsimpleoptions
PERSISTED_RECORD_U_Q_KEY_REF	persistedrecorduqkeyref
PERSISTED_RECORD_U_Q_KEY_REF_TYPE	persistedrecorduqkeyreftype
PHONE_CALL	phonecall
PLUG_IN_TYPE	plugintype
PLUG_IN_TYPE_IMPL	plugintypeimpl
PORTLET	portlet
PORTLET_DEPLOYMENT	portletdeployment
PRICE_LEVEL	pricelevel
PRICING	pricing
PRICING_GROUP	pricinggroup
PRIMARY_RECORD	primaryrecord
PROJECT_TASK	projecttask
PROJECT_TEMPLATE	projecttemplate
PROMOTIONS_PLUGIN	promotionsplugin
PROMOTION_CODE	promotioncode
PUBLISHED_SAVED_SEARCH	publishedsavedsearch
PURCHASE_ORDER	purchaseorder
PURCHASE_REQUISITION	purchaserequisition
QUANTITY_PRICING_SCHEDULE	quantitypricingschedule
QUOTA	quota
RECENT_RECORD	recentrecord
RECORD_SERVICE_TEST_RECORD	recordservicetestrecord
RECORD_TYPE	recordtype
RECORD_WITH_HIERARCHY_RELATIONSHIP	recordwithhierarchyrelationship
REDIRECT	redirect

Enum Value	Sets Query.type Property To
REGION	region
RELATIONSHIP_DISPLAY_INACTIVE	relationshipdisplayinactive
RELATIONSHIP_SELECT_EMPLOYEE_RECORD	relationshipselectemployeeerecord
REPORT_DEFINITION	reportdefinition
REQUEST_LEVEL_RECORD1	requestlevelrecord1
REQUEST_LEVEL_RECORD2	requestlevelrecord2
RESOURCE	resource
RESTLET	restlet
RESTLET_DEPLOYMENT	restletdeployment
RESTRICTIONS_ONCE_REMOVED	restrictionsonceremoved
RESTRICTIONS_TWICE_REMOVED	restrictionstwiceremoved
RESTRICTION_ANNOTATION_TEST_RECORD	restrictionannotationtestrecord
RESTRICTION_TEST_RECORD	restrictiontestrecord
RETURN_AUTHORIZATION	returnauthorization
REV_REC_SCHEDULE	revrecschedule
REV_REC_TEMPLATE	revrectemplate
ROLE	role
RSTR_ALT_LOCATION	rstraltlocation
RSTR_LOCATION	rstrlocation
RSTR_RECORD	rstrrecord
SALES	sales
SALES_ORDER	salesorder
SALES_READINESS	salesreadiness
SALES_ROLE	salesrole
SALES_TAX_ITEM	salestaxitem
SALES_TERRITORY	salesterritory
SALES_TRANSACTION	salestransaction
SAMPLE_RECORD	samplerrecord
SCHEDULED_SCRIPT	scheduledscript
SCHEDULED_SCRIPT_DEPLOYMENT	scheduledscriptdeployment
SCHEDULED_SCRIPT_INSTANCE	scheduledscriptinstance
SCRIPT	script
SCRIPTING_TEST_RECORD	scriptingtestrecord
SCRIPTING_TEST_RECORD_SUBRECORD2_TARGET	scriptingtestrecordsubrecord2target

Enum Value	Sets Query.type Property To
SCRIPTING_TEST_RECORD_SUBRECORD2_TARGET2	scriptingtestrecordsubrecord2target2
SCRIPTING_TEST_RECORD_SUBRECORD3_TARGET	scriptingtestrecordsubrecord3target
SCRIPTING_TEST_RECORD_SUBRECORD3_TARGET2	scriptingtestrecordsubrecord3target2
SCRIPTING_TEST_RECORD_SUBRECORD4_TARGET	scriptingtestrecordsubrecord4target
SCRIPTING_TEST_RECORD_SUBRECORD4_TARGET2	scriptingtestrecordsubrecord4target2
SCRIPTING_TEST_RECORD_SUBRECORD_TARGET	scriptingtestrecordsubrecordtarget
SCRIPTING_TEST_RECORD_SUBRECORD_TARGET2	scriptingtestrecordsubrecordtarget2
SCRIPTING_TEST_RECORD_TARGET	scriptingtestrecordtarget
SCRIPTING_TEST_RECORD_TARGET2	scriptingtestrecordtarget2
SCRIPT_DEPLOYMENT	scriptdeployment
SCRIPT_NOTE	scriptnote
SCRIPT_RECORD_TYPE	scriptrecordtype
SCRIP_INH_TEST_RECORD1	scripinhtestrecord1
SCRIP_INH_TEST_RECORD2	scripinhtestrecord2
SCRIP_INH_TEST_RECORD3	scripinhtestrecord3
SCRIP_INH_TEST_RECORD4	scripinhtestrecord4
SEARCH_CAMPAIGN	searchcampaign
SEARCH_SCHEDULE	searchschedule
SEARCH_URL_TEST_SOURCE_RECORD	searchurltestsourcerecord
SEARCH_URL_TEST_TARGET_RECORD	searchurltesttargetrecord
SELECT_OPTIONS_SOURCE_RECORD	selectoptionssourcerecord
SERVICE_PURCHASE_ITEM	servicepurchaseitem
SERVICE_RESALE_ITEM	serviceresaleitem
SERVICE_SALE_ITEM	servicesaleitem
SHIPPING_PACKAGE	shippingpackage
SHIPPING_PARTNERS_PLUGIN	shippingpartnersplugin
SHIP_ITEM	shipitem
SHOPPING_CART	shoppingcart
SIMPLE_RECORD	simplerecord
SIMPLE_RECORD_LOCATION	simplerecordlocation
SITE_CATEGORY	sitecategory
SLAVE	slave
SLAVE_EMPTY_EXPRESSION	slaveemptyexpression
SLAVE_FEATURE	slavefeature

Enum Value	Sets Query.type Property To
SLAVE_MASTER_PERMISSION	slavemasterpermission
SLAVE_PERMISSION	slavepermission
SLAVE_TARGET_PROPERTY	slavetargetproperty
SLAVE_VALID_EXPRESSION	slavevalidexpression
SOLUTION	solution
SORT_BASE_RECORD	sortbaserecord
SORT_RECORD	sortrecord
SORT_RELATED_RECORD	sortrelatedrecord
STATIC_LIST_RECORD	staticlistrecord
STATIC_OPTIONS_FIELD_TEST_RECORD	staticoptionsfieldtestrecord
STATIC_OPTIONS_VALUE	staticoptionsvalue
STORE_TAB	storetab
STUDENT_RECORD	studentrecord
SUBLIST	sublist
SUBSIDIARY	subsidiary
SUBTOTAL_ITEM	subtotalitem
SUB_SELECT_GROUP_RECORD	subselectgrouprecord
SUITELET	suitelet
SUITELET_DEPLOYMENT	suiteletdeployment
SUITE_SCRIPT_DETAIL	suitescriptdetail
SUPPORT_CASE	supportcase
SUPPORT_CASE_ISSUE	supportcaseissue
SUPPORT_CASE_ORIGIN	supportcaseorigin
SUPPORT_CASE_PRIORITY	supportcasepriority
SUPPORT_CASE_STATUS	supportcasestatus
SUPPORT_CASE_TYPE	supportcasetype
SUPPORT_TERRITORY	supportterritory
SYSTEM_EMAIL_TEMPLATE	systememailtemplate
SYSTEM_JOURNAL	systemjournal
SYSTEM_NOTE	systemnote
SYSTEM_NOTE_FIELD	systemnotefield
TABLE_CONDITION_TEST_RECORD	tableconditiontestrecord
TASK	task
TASK_ITEM_STATUS	taskitemstatus

Enum Value	Sets Query.type Property To
TAX_CALCULATION_PLUGIN	taxcalculationplugin
TAX_ITEM_TAX_GROUP	taxitemtaxgroup
TAX_PERIOD	taxperiod
TAX_TYPE	taxtype
TERM	term
TESTDOAGGREGATEDOSUBTYPE	testdoaggregatedosubtype
TESTDOAGGREGATERESTRICTIONRECORD	testdoaggregaterestrictionrecord
TEST_COMPOSED_RECORD1	testcomposedrecord1
TEST_COMPOSED_RECORD2	testcomposedrecord2
TEST_COMPOSED_RECORD3	testcomposedrecord3
TEST_CONFIGURABLE_RECORD	testconfigurablerecord
TEST_DO_AGGREGATE_RECORD	testdoaggregaterecord
TEST_EXPOSURE_RECORD	testexposurerecord
TEST_FEATURE_RECORD	testfeaturerecord
TEST_FULL_RECORD	testfullrecord
TEST_MACROS_UMD_RECORD	testmacrosumdrecord
TEST_MULTI_TABLE_PERSISTENCE_RECORD	testmultitablepersistencerecord
TEST_NEW_URLS_RECORD	testnewurlsrecord
TEST_NEW_URLS_TARGET_RECORD	testnewurlstargetrecord
TEST_NEW_URLS_UNSUPPORTED_RECORD	testnewurlsunsupportedrecord
TEST_NEXT_STANDARD_RECORD	testnextstandardrecord
TEST_PLUGIN	testplugin
TEST_PRIMARY_RECORD_FOR_RELATIONSHIPS	testprimaryrecordforrelationships
TEST_RECORD	testrecord
TEST_RECORD_ACTION_RECORD	testrecordactionrecord
TEST_RECORD_UNIQUE_KEY	testrecorduniquekey
TEST_RECORD_WITHOUT_LABEL	testrecordwithoutlabel
TEST_RECORD_WITH_DISABLED_RECORD_SORT_FIELDS	testrecordwithdisabledrecordsortfields
TEST_RECORD_WITH_SORT_FIELDS	testrecordwithsortfields
TEST_REGRESSION_RECORD	testregressionrecord
TEST_RELATED_PROPERTY	testrelatedproperty
TEST_SECURED_RECORD	testsecuredrecord
TEST_SIMPLE_PERSISTENCE_RECORD	testsimplepersistencerecord
TEST_SIMPLE_PERSISTENCE_SELECT_SIDE_RECORD	testsimplepersistenceseselectsiderecord

Enum Value	Sets Query.type Property To
TEST_SLAVING_RATE_FIELD_RECORD	testslavingratefieldrecord
TEST_SLAVING_RECORD	testslavingrecord
TEST_SLAVING_RECORD_OPTIMIZED	testslavingrecordoptimized
TEST_SOURCING_MASTER_FIELD_ANNOTATION_MASTER	testsourcingmasterfieldannotationmaster
TEST_SOURCING_MASTER_FIELD_ANNOTATION_RECORD	testsourcingmasterfieldannotationrecord
TEST_SOURCING_OPTIONS_CONDITION_MASTER	testsourcingoptionsconditionmaster
TEST_SOURCING_OPTIONS_CONDITION_RECORD	testsourcingoptionsconditionrecord
TEST_SOURCING_OPTIONS_CONDITION_TARGET	testsourcingoptionsconditiontarget
TEST_SOURCING_SUBLIST_FIELD_ANNOTATION_MASTER	testsourcingsublistfieldannotationmaster
TEST_SOURCING_SUBLIST_FIELD_ANNOTATION_RECORD	testsourcingsublistfieldannotationrecord
TEST_SOURCING_VALUE_RATE_COL_MASTER	testsourcingvalueratecolmaster
TEST_SOURCING_VALUE_RATE_COL_RECORD	testsourcingvalueratecolrecord
TEST_STANDARD_RECORD	teststandardrecord
TEST_TRANSACTION	testtransaction
TIME_BILL	timebill
TOPIC	topic
TRACKING_NUMBER	trackingnumber
TRANSACTION	transaction
TRANSACTION_ADDRESSBOOK	transactionaddressbook
TRANSACTION_BILLING_ADDRESSBOOK	transactionbillingaddressbook
TRANSACTION_NUMBERING_AUDIT_LOG	transactionnumberingauditlog
TRANSACTION_RETURN_ADDRESSBOOK	transactionreturnaddressbook
TRANSACTION_SHIPPING_ADDRESSBOOK	transactionshippingaddressbook
TRANSFER	transfer
TRANSFER_ORDER	transferorder
TWO_FACTOR_DEVICE	twofactordevice
TYPE_FIELD_PARENT_RECORD	typefieldparentrecord
TYPE_FIELD_RECORD	typefieldrecord
UMD_FIELD	umdfield
UNDELIVERED_EMAIL	undeliveredemail
UNIFICATION_TEST	unificationtest
USER_EVENT_SCRIPT	usereventscript
USER_EVENT_SCRIPT_DEPLOYMENT	usereventscriptdeployment
USRCATEGORY	usrcategory

Enum Value	Sets Query.type Property To
USRSAVEDSEARCH	usrsavedsearch
USR_ANALYTICAL	usranalytical
USR_AUDITING_SOURCE_RECORD	usrauditingsourcerecord
USR_AUDIT_LOG	usrauditlog
USR_CHANNEL_AG_BTH_ROOT	usrchannelagbthroot
USR_CHANNEL_AG_BTH_ROOT_SUB_TYPE	usrchannelagbthrootsubtype
USR_CHANNEL_AG_BTH_SEARCH_MTM_ROOT	usrchannelagbthsearchmtmroot
USR_CHANNEL_AG_BTH_SEARCH_MTM_SUB_TYPE	usrchannelagbthsearchmtmsubtype
USR_CHANNEL_AG_BTH_SEARCH_MTO_ROOT	usrchannelagbthsearchmtoroot
USR_CHANNEL_AG_BTH_SEARCH_MTO_SUB_TYPE	usrchannelagbthsearchmtosubtype
USR_CHANNEL_AG_SRC_ROOT	usrchannelagsrcroot
USR_CHANNEL_AG_SRC_ROOT_SUB_TYPE	usrchannelagsrcrootsubtype
USR_CHANNEL_AG_SRC_SEARCH_MTM_PRIMARY	usrchannelagsrcsearchmtmprimary
USR_CHANNEL_AG_SRC_SEARCH_MTO_PRIMARY	usrchannelagsrcsearchmtopprimary
USR_CHANNEL_AG_TGT_ROOT	usrchannelagtgtroot
USR_CHANNEL_AG_TGT_SEARCH_MTM_ROOT	usrchannelagtgtsearchmtmroot
USR_CHANNEL_AG_TGT_SEARCH_MTM_SUB_TYPE	usrchannelagtgtsearchmtmsubtype
USR_CHANNEL_AG_TGT_SEARCH_MTO_ROOT	usrchannelagtgtsearchmtoroot
USR_CHANNEL_AG_TGT_SEARCH_MTO_SUB_TYPE	usrchannelagtgtsearchmtosubtype
USR_CHANNEL_STD_ROOT	usrchannelstdroot
USR_CHANNEL_STD_SEARCH_MTM_PRIMARY	usrchannelstdsearchmtmprimary
USR_CHANNEL_STD_SEARCH_MTO_PRIMARY	usrchannelstdsearchmtopprimary
USR_EXECUTION_LOG	usrexecutionlog
USR_EXPOSE_EXTERNAL	usrexposeexternal
USR_EXPOSE_IMPORTANT	usrexposeimportant
USR_EXPOSE_INTNL_FLD_PLAIN_AG_TGT_PLAIN_MTO_ROOT	usrexposeintnlfldplainagtgtplainmtoroot
USR_EXPOSE_INTNL_FLD_PLAIN_AG_TGT_PLAIN_MTO_SUB_TYPE	usrexposeintnlfldplainagtgtplainmtosubtype
USR_EXPOSE_INTNL_FLD_PLAIN_AG_TGT_ROOT	usrexposeintnlfldplainagtgtroot
USR_EXPOSE_INTNL_FLD_PLAIN_STD_N_VAL_MTO_PRIMARY	usrexposeintnlfldplainstdnvalmtopprimary
USR_EXPOSE_INTNL_FLD_PLAIN_STD_ROOT	usrexposeintnlfldplainstdroot
USR_EXPOSE_PLAIN_FLD_INTNL_AG_BTH_N_VAL_MTO_ROOT	usrexposeplainfldintnlagbthnvalmtoroot
USR_EXPOSE_PLAIN_FLD_INTNL_AG_BTH_N_VAL_MTO_SUB_TYPE	usrexposeplainfldintnlagbthnvalmtosubtype

Enum Value	Sets Query.type Property To
USR_EXPOSE_PLAIN_FLD_INTNL_AG_BTH_PLAIN_MTO_ROOT	usrexposeplainfldintnlagbthplainmtoroot
USR_EXPOSE_PLAIN_FLD_INTNL_AG_BTH_PLAIN_MTO_SUB_TYPE	usrexposeplainfldintnlagbthplainmtosubtype
USR_EXPOSE_PLAIN_FLD_INTNL_AG_SRC_N_VAL_MTO_PRIMARY	usrexposeplainfldintnlagsrcnvalmtopprimary
USR_EXPOSE_PLAIN_FLD_INTNL_AG_SRC_PLAIN_MTO_PRIMARY	usrexposeplainfldintnlagsrcplainmtopprimary
USR_EXPOSE_PLAIN_FLD_INTNL_AG_TGT_N_VAL_MTO_ROOT	usrexposeplainfldintnlagtgtntvalmtoroot
USR_EXPOSE_PLAIN_FLD_INTNL_AG_TGT_N_VAL_MTO_SUB_TYPE	usrexposeplainfldintnlagtgtntvalmtosubtype
USR_EXPOSE_PLAIN_FLD_INTNL_AG_TGT_PLAIN_MTO_ROOT	usrexposeplainfldintnlagtgtplainmtoroot
USR_EXPOSE_PLAIN_FLD_INTNL_AG_TGT_PLAIN_MTO_SUB_TYPE	usrexposeplainfldintnlagtgtplainmtosubtype
USR_EXPOSE_PLAIN_FLD_INTNL_STD_N_VAL_MTM_PRIMARY	usrexposeplainfldintnlstdnvalmtmprimary
USR_EXPOSE_PLAIN_FLD_INTNL_STD_N_VAL_MTO_PRIMARY	usrexposeplainfldintnlstdnvalmtopprimary
USR_EXPOSE_PLAIN_FLD_INTNL_STD_PLAIN_MTM_PRIMARY	usrexposeplainfldintnlstdplainmtmprimary
USR_EXPOSE_PLAIN_FLD_INTNL_STD_PLAIN_MTO_PRIMARY	usrexposeplainfldintnlstdplainmtopprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_N_VAL_MTO_ROOT	usrexposeplainfldplainagbthnvalmtoroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_N_VAL_MTO_SUB_TYPE	usrexposeplainfldplainagbthnvalmtosubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_PLAIN_MTO_ROOT	usrexposeplainfldplainagbthplainmtoroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_PLAIN_MTO_SUB_TYPE	usrexposeplainfldplainagbthplainmtosubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_ROOT	usrexposeplainfldplainagbthroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_BTH_SUB_TYPE	usrexposeplainfldplainagbthsubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_N_VAL_MTM_PRIMARY	usrexposeplainfldplainagsrcnvalmtmprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_N_VAL_MTO_PRIMARY	usrexposeplainfldplainagsrcnvalmtopprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_PLAIN_MTM_PRIMARY	usrexposeplainfldplainagsrcplainmtmprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_PLAIN_MTO_PRIMARY	usrexposeplainfldplainagsrcplainmtopprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_ROOT	usrexposeplainfldplainagsrcroot

Enum Value	Sets Query.type Property To
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_SRC_SUB_TYPE	usrexposeplainfldplainagsrcsubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_N_VAL_MTM_ROOT	usrexposeplainfldplainagtgtinvalmtmroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_N_VAL_MTM_SUB_TYPE	usrexposeplainfldplainagtgtinvalmtmsubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_N_VAL_MTO_ROOT	usrexposeplainfldplainagtgtinvalmtoroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_N_VAL_MTO_SUB_TYPE	usrexposeplainfldplainagtgtinvalmtosubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_PLAIN_MTM_ROOT	usrexposeplainfldplainagtgtplainmtmroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_PLAIN_MTM_SUB_TYPE	usrexposeplainfldplainagtgtplainmtmsubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_PLAIN_MTO_ROOT	usrexposeplainfldplainagtgtplainmtoroot
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_PLAIN_MTO_SUB_TYPE	usrexposeplainfldplainagtgtplainmtosubtype
USR_EXPOSE_PLAIN_FLD_PLAIN_AG_TGT_ROOT	usrexposeplainfldplainagtgtroot
USR_EXPOSE_PLAIN_FLD_PLAIN_STD_N_VAL_MTM_PRIMARY	usrexposeplainfldplainstdnvalmtmprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_STD_N_VAL_MTO_PRIMARY	usrexposeplainfldplainstdnvalmtopprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_STD_PLAIN_MTM_PRIMARY	usrexposeplainfldplainstdplainmtmprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_STD_PLAIN_MTO_PRIMARY	usrexposeplainfldplainstdplainmtopprimary
USR_EXPOSE_PLAIN_FLD_PLAIN_STD_ROOT	usrexposeplainfldplainstdroot
USR_FEATURE_AG_BTH_ROOT	usrfeatureagbthroot
USR_FEATURE_AG_BTH_ROOT_SUB_TYPE	usrfeatureagbthrootsubtype
USR_FEATURE_AG_SRC_ROOT	usrfeatureagsrcroot
USR_FEATURE_AG_SRC_ROOT_SUB_TYPE	usrfeatureagsrcrootsubtype
USR_FEATURE_AG_TGT_ROOT	usrfeatureagtgtroot
USR_FEATURE_CSM_DEFAULT_COLUMNS_RECORD	usrfeaturecsmdefaultcolumnsrecord
USR_FEATURE_CSM_IMPORTANT_JOIN_RECORD	usrfeaturecsmimportantjoinrecord
USR_FEATURE_CSM_INHERITANCE_RECORD	usrfeaturecsminheritancerecord
USR_FEATURE_CSM_USAGE_SPECIFIC_RECORD	usrfeaturecsmusagespecificrecord
USR_FEATURE_STD_ROOT	usrfeaturestdroot
USR_NON_SYSTEM_RECORD	usrnonsystemrecord
USR_PERMISSION_AG_BTH_DENIED_MTM_ROOT	usrpermissionagbthdeniedmtmroot
USR_PERMISSION_AG_BTH_DENIED_MTM_SUB_TYPE	usrpermissionagbthdeniedmtmsubtype
USR_PERMISSION_AG_BTH_DENIED_MTO_ROOT	usrpermissionagbthdeniedmtoroot

Enum Value	Sets Query.type Property To
USR_PERMISSION_AG_BTH_DENIED_MTO_SUB_TYPE	usrpermissionagbthdeniedmtosubtype
USR_PERMISSION_AG_BTH_GRANTED_MTM_ROOT	usrpermissionagbthgrantedmtmroot
USR_PERMISSION_AG_BTH_GRANTED_MTM_SUB_TYPE	usrpermissionagbthgrantedmtmsubtype
USR_PERMISSION_AG_BTH_GRANTED_MTO_ROOT	usrpermissionagbthgrantedmtoroot
USR_PERMISSION_AG_BTH_GRANTED_MTO_SUB_TYPE	usrpermissionagbthgrantedmtosubtype
USR_PERMISSION_AG_BTH_ROOT	usrpermissionagbthroot
USR_PERMISSION_AG_BTH_ROOT_SUB_TYPE	usrpermissionagbthrootsubtype
USR_PERMISSION_AG_SRC_DENIED_MTM_PRIMARY	usrpermissionagsrcdeniedmtmprimary
USR_PERMISSION_AG_SRC_DENIED_MTO_PRIMARY	usrpermissionagsrcdeniedmtopprimary
USR_PERMISSION_AG_SRC_GRANTED_MTM_PRIMARY	usrpermissionagsrcgrantedmtmprimary
USR_PERMISSION_AG_SRC_GRANTED_MTO_PRIMARY	usrpermissionagsrcgrantedmtopprimary
USR_PERMISSION_AG_SRC_ROOT	usrpermissionagsrcroot
USR_PERMISSION_AG_SRC_ROOT_SUB_TYPE	usrpermissionagsrcrootsubtype
USR_PERMISSION_AG_TGT_DENIED_MTM_ROOT	usrpermissionagtgt deniedmtmroot
USR_PERMISSION_AG_TGT_DENIED_MTM_SUB_TYPE	usrpermissionagtgt deniedmtmsubtype
USR_PERMISSION_AG_TGT_DENIED_MTO_ROOT	usrpermissionagtgt deniedmtoroot
USR_PERMISSION_AG_TGT_DENIED_MTO_SUB_TYPE	usrpermissionagtgt deniedmtosubtype
USR_PERMISSION_AG_TGT_GRANTED_MTM_ROOT	usrpermissionagtgt grantedmtmroot
USR_PERMISSION_AG_TGT_GRANTED_MTM_SUB_TYPE	usrpermissionagtgt grantedmtmsubtype
USR_PERMISSION_AG_TGT_GRANTED_MTO_ROOT	usrpermissionagtgt grantedmtoroot
USR_PERMISSION_AG_TGT_GRANTED_MTO_SUB_TYPE	usrpermissionagtgt grantedmtosubtype
USR_PERMISSION_AG_TGT_ROOT	usrpermissionagtgtroot
USR_PERMISSION_STD_DENIED_MTM_PRIMARY	usrpermissionstd deniedmtmprimary
USR_PERMISSION_STD_DENIED_MTO_PRIMARY	usrpermissionstd deniedmtopprimary
USR_PERMISSION_STD_GRANTED_MTM_PRIMARY	usrpermissionstd grantedmtmprimary
USR_PERMISSION_STD_GRANTED_MTO_PRIMARY	usrpermissionstd grantedmtopprimary
USR_PERMISSION_STD_ROOT	usrpermissionstdroot
USR_POLYMORPHIC_CHILD_ONE_RECORD	usrpolymorphicchildonerecord
USR_POLYMORPHIC_CHILD_TWO_RECORD	usrpolymorphicchildtworecord
USR_POLYMORPHIC_JOIN_TEST_RECORD	usrpolymorphicjointestrecord
USR_TARGET_PROPERTIES_GROUP_BY_TARGET_RECORD	usrtargetpropertiesgroupbytargetrecord
USR_TARGET_PROPERTIES_MTO2_TARGET_RECORD	usrtargetpropertiesmto2targetrecord
USR_TARGET_PROPERTIES_MTO_TARGET_RECORD	usrtargetpropertiesmtotargetrecord
USR_TARGET_PROPERTIES_ROOT_RECORD	usrtargetpropertiesrootrecord

Enum Value	Sets Query.type Property To
USR_UNIVERSAL	usruniversal
VENDOR	vendor
VENDOR_BILL	vendorbill
VENDOR_CATEGORY	vendorcategory
VENDOR_CREDIT	vendorcredit
VENDOR_PAYMENT	vendorpayment
VENDOR_SUBSIDIARY_RELATIONSHIP	vendorsubsubsidiaryrelationship
WEBAPP	webapp
WEB_SITE	website
WIN_LOSS_REASON	winlossreason
WORKFLOW_ACTION_SCRIPT	workflowactionscrip
WORKFLOW_ACTION_SCRIPT_DEPLOYMENT	workflowactionscripdeployment
WORKPLACE	workplace
WORK_CALENDAR	workcalendar

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/query Module Script Samples](#).

```
var search = query.create({
    type: query.Type.CUSTOMER
});

var salesrep = search.autoJoin({
    fieldId: 'salesrep'
});

var cond1 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 107
});
var cond2 = search.createCondition({
    fieldId: 'id',
    operator: query.Operator.EQUAL,
    values: 2647
});
var cond3 = salesrep.createCondition({
    fieldId: 'email',
    operator: query.Operator.START_WITH_NOT,
    values: 'foo'
});
search.condition = search.and(
    cond3, search.or(cond1, cond2)
```

```
);

var resultSet = search.run();
```

N/record Module

Load the record module to work with NetSuite records. You can use this module to create, delete, copy, load, or make changes to a record.

SuiteScript supports working with standard NetSuite records and with instances of custom record types. Supported standard record types are described in the [SuiteScript Records Browser](#). Refer also to [SuiteScript Supported Records](#). For help working with an instance of a custom record type, see the help topic [Custom Record](#).



Important: SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). The NetSuite UI should only be accessed with SuiteScript APIs.

- [N/record Module Members](#)
- [Column Object Members](#)
- [Field Object Members](#)
- [Macro Object Members](#)
- [Record Object Members](#)
- [Sublist Object Members](#)
- [N/record Module Script Samples](#)
- [N/record Default Values](#)

N/record Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	record.Column	Object	Client and server-side scripts	Encapsulates a column of a sublist on a standard or custom record.
	record.Field	Object	Client and server-side scripts	Encapsulates a body or sublist field on a standard or custom record.
	record.Macro	Object	Client and server-side scripts	Encapsulates a NetSuite record macro.
	Plain JavaScript Object	Object	Client and server-side scripts	A plain JavaScript object of record macros available for a record type. This object is returned by Record.getMacros(options) .
	record.Record	Object	Client and server-side scripts	Encapsulates a NetSuite record.
	record.Sublist	Object	Client and server-side scripts	Encapsulates a sublist on a standard or custom record.
Method	record.attach(options)	void	Client and server-side scripts	Attaches a record to another record.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	record.attach.promise(options)	Promise	Client scripts	Attaches a record asynchronously to another record.
	record.copy(options)	record.Record	Client and server-side scripts	Creates a new record by copying an existing record in NetSuite.
	record.copy.promise(options)	Promise	Client scripts	Creates a new record asynchronously by copying an existing record in NetSuite.
	record.create(options)	record.Record	Client and server-side scripts	Creates a new record.
	record.create.promise(options)	Promise	Client scripts	Creates a new record asynchronously.
	record.delete(options)	number	Client and server-side scripts	Deletes a record.
	record.delete.promise(options)	Promise	Client scripts	Deletes a record asynchronously.
	record.detach(options)	void	Client and server-side scripts	Detaches a record from another record.
	record.detach.promise(options)	Promise	Client scripts	Detaches a record from another record asynchronously.
	record.submitFields.promise(options)	Promise	Client scripts	Updates and submits one or more body fields asynchronously on an existing record in NetSuite, and returns the internal ID of the parent record.
	record.transform(options)	record.Record	Client and server-side scripts	Transforms a record from one type into another, using data from an existing record.
	record.transform.promise(options)	Promise	Client scripts	Transforms a record from one type into another asynchronously, using data from an existing record.
Enum	record.Type	enum	Client and server-side scripts	Enumeration that holds the string values for supported standard record types.

Column Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Column.id	string (read-only)	Client and server-side scripts	Returns the internal ID of the column.
	Column.label	string (read-only)	Client and server-side scripts	Returns the UI label for the column.
	Column.sublistId	string (read-only)	Client and server-side scripts	Returns the internal ID of the standard or custom sublist that contains the column.
	Column.type	string (read-only)	Client and server-side scripts	Returns the column type.

Field Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Field.getSelectOptions(options)	array	Client and server-side scripts	Returns an array of available options on a standard or custom select, multiselect, or radio field as key-value pairs. Only the first 1,000 available options are returned.
Property	Field.label	string (read-only)	Client and server-side scripts	Returns the UI label for a standard or custom field body or sublist field.
	Field.id	string (read-only)	Client and server-side scripts	Returns the internal ID of a standard or custom body or sublist field.
	Field.type	string (read-only)	Client and server-side scripts	Returns the type of a body or sublist field.
	Field.isMandatory	boolean <code>true</code> <code>false</code>	Client and server-side scripts	Returns <code>true</code> if the standard or custom field is mandatory on the record form, or <code>false</code> otherwise.
	Field.sublistId	string (read-only)	Client and server-side scripts	Returns the ID of the sublist associated with the specified sublist field.
	Field.isDisplay	boolean <code>true</code> <code>false</code>	Client and server-side scripts	Returns <code>true</code> if the field is visible on the record form, or <code>false</code> if it is not.

Macro Object Members

The following members are called on the [record.Macro](#) object. For information about record macros, see [Overview of Record Action and Macro APIs](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Macro.execute(options)	Object	Client and server-side scripts	Performs a macro operation and returns its result in an object.
	Macro.execute.promise(options)	Promise	Client scripts	Performs a macro operation asynchronously.
	Macro(options)	Object	Client and server-side scripts	Performs a macro operation and returns its result in an object.
	Macro.promise(options)	Promise	Client scripts	Performs a macro operation asynchronously.
Property	Macro.id	string	Client and server-side scripts	The ID of the macro. For a list of macro IDs, see Supported Record Macros
	Macro.label	string	Client and server-side scripts	The macro label.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Macro.description	string	Client and server-side scripts	The macro description.
	Macro.attributes	Object	Client and server-side scripts	The macro defined attributes.

Record Object Members

The following members are called on the `record.Record` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Record.cancelLine(options)	record.Record	Client and server-side scripts	Cancels the currently selected line on a sublist.
	Record.commitLine(options)	record.Record	Client and server-side scripts	Commits the currently selected line on a sublist.
	Record.executeMacro(options)	Object	Client and server-side scripts	Performs macro operation and returns its result in a plain JavaScript object.
	Record.getMacros(options)	Object	Client and server-side scripts	Provides a plain JavaScript object that contains macro objects defined for a record type, indexed by the Macro ID.
	Record.findMatrixSublistLineWithValue(number, value, options)	number	Client and server-side scripts	Returns the line number of the first instance where a specified value is found in a specified column of the matrix.
	Record.findSublistLineWithValue(number, value, options)	number	Client and server-side scripts	Returns the line number for the first occurrence of a field value in a sublist.
	Record.getCurrentMatrixSublistValue(number, options)	number Date string array boolean true false	Client and server-side scripts	Gets the value for the currently selected line in the matrix.
	Record.getCurrentSublistField(options)	record.Field	Client and server-side scripts	Returns a field object from a sublist.
	Record.getCurrentSublistIndex(options)	number	Client and server-side scripts	Returns the line number of the currently selected line.
	Record.getCurrentSublistSubrecord(options)	record.Record	Client and server-side scripts	Gets the subrecord for the associated sublist field on the current line.
	Record.getCurrentSublistText(options)	string	Client and server-side scripts	Returns a text representation of the field value in the currently selected line.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Record.getCurrentSublistValue(options)	number Date string array boolean true false	Client and server-side scripts	Returns the value of a sublist field on the currently selected sublist line.
	Record.getField(options)	record.Field	Client and server-side scripts	Returns a field object from a record.
	Record.getFields()	string[]	Client and server-side scripts	Returns the body field names (internal ids) of all the fields in the record, including machine header field and matrix header fields.
	Record.getLineCount(options)	number	Client and server-side scripts	Returns the number of lines in a sublist.
	Record.getMacro(options)	record.Macro	Client and server-side scripts	Provides a macro to execute.
	Record.getMacros(options)	Object	Client and server-side scripts	Provides a plain JavaScript object that contains macro objects defined for a record type, indexed by the Macro ID.
	Record.getMatrixHeaderCount(options)	number	Client and server-side scripts	Returns the number of columns for the specified matrix.
	Record.getMatrixHeaderField(options)	record.Field	Client and server-side scripts	Gets the field for the specified header in the matrix.
	Record.getMatrixHeaderValue(options)	number Date string array boolean true false	Client and server-side scripts	Gets the value for the associated header in the matrix.
	Record.getMatrixSublistField(options)	record.Field	Client and server-side scripts	Gets the field for the specified sublist in the matrix.
	Record.getMatrixSublistValue(options)	number Date string array boolean true false	Client and server-side scripts	Gets the value for the associated field in the matrix.
	Record.getSublist(options)	record.Sublist	Client and server-side scripts	Returns the specified sublist.
	Record.getSublists()	string[]	Client and server-side scripts	Returns all the names of all the sublists.
	Record.getSublistField(options)	record.Field	Client and server-side scripts	Returns a field object from a sublist.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>Record.getSublistFields(options)</code>	<code>string[]</code>	Client and server-side scripts	Returns all the field names in a sublist.
	<code>Record.getSublistSubrecord(options)</code>	<code>record.Record</code>	Client and server-side scripts	Gets the subrecord associated with a sublist field. (standard mode only)
	<code>Record.getSublistText(options)</code>	<code>string</code>	Client and server-side scripts	Returns the value of a sublist field in a text representation.
	<code>Record.getSublistValue(options)</code>	<code>number Date string array boolean true false</code>	Client and server-side scripts	Returns the value of a sublist field.
	<code>Record.getSubrecord(options)</code>	<code>record.Record</code>	Client and server-side scripts	Gets the subrecord for the associated field.
	<code>Record.getText(options)</code>	<code>string</code>	Client and server-side scripts	Returns the text representation of a field value.
	<code>Record.getValue(options)</code>	<code>number Date string array boolean true false</code>	Client and server-side scripts	Returns the value of a field.
	<code>Record.hasCurrentSublistSubrecord(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns a value indicating whether the associated sublist field has a subrecord on the current line.
	<code>Record.hasSublistSubrecord(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns a value indicating whether the associated sublist field contains a subrecord.
	<code>Record.hasSubrecord(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns a value indicating whether the field contains a subrecord.
	<code>Record.insertLine(options)</code>	<code>record.Record</code>	Client and server-side scripts	Inserts a sublist line.
	<code>Record.removeCurrentSublistSubrecord(options)</code>	<code>record.Record</code>	Client and server-side scripts	Removes the subrecord for the associated sublist field on the current line.
	<code>Record.removeLine(options)</code>	<code>record.Record</code>	Client and server-side scripts	Removes a sublist line.
	<code>Record.removeSublistSubrecord(options)</code>	<code>record.Record</code>	Client and server-side scripts	Removes the subrecord for the associated sublist field. (standard mode only)
	<code>Record.removeSubrecord(options)</code>	<code>record.Record</code>	Client and server-side scripts	Removes the subrecord for the associated field.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Record.save(options)	number	Client and server-side scripts	Submits a new record or saves edits to an existing record. This method is not available to subrecords.
	Record.save.promise(options)	number	Client scripts	Submits a new record asynchronously or saves edits to an existing record asynchronously. This method is not available to subrecords.
	Record.selectLine(options)	record.Record	Client and server-side scripts	Selects an existing line in a sublist.
	Record.selectNewLine(options)	record.Record	Client and server-side scripts	Selects a new line at the end of a sublist.
	Record.setCurrentMatrixSublistValue(options)	record.Record	Client and server-side scripts	Sets the value for the line currently selected in the matrix.
	Record.setCurrentSublistText(options)	record.Record	Client and server-side scripts	Sets the value for the field in the currently selected line by a text representation.
	Record.setCurrentSublistValue(options)	record.Record	Client and server-side scripts	Sets the value for the field in the currently selected line.
	Record.setMatrixHeaderValue(options)	record.Record	Client and server-side scripts	Sets the value for the associated header in the matrix.
	Record.setMatrixSublistValue(options)	record.Record	Client and server-side scripts	Sets the value for the associated field in the matrix.
	Record.setSublistText(options)	record.Record	Client and server-side scripts	Sets the value of a sublist field by a text representation. (standard mode only)
	Record.setSublistValue(options)	record.Record	Client and server-side scripts	Sets the value of a sublist field. (standard mode only)
	Record.setText(options)	record.Record	Client and server-side scripts	Sets the value of the field by a text representation.
	Record.setValue(options)	record.Record	Client and server-side scripts	Sets the value of a field.
Property	Record.id	number (read-only)	Client and server-side scripts	The internal ID of a specific record. This property is not available to subrecords.
	Record.isDynamic	boolean (read-only)	Client and server-side scripts	Indicates whether the record is in dynamic or standard mode.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Record.type	string (read-only)	Client and server-side scripts	The record type. This property is not available to subrecords.

Sublist Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Sublist.getColumn(options)	record.Column	Client and server-side scripts	Returns a column in the sublist.
Property	Sublist.id	string (read-only)	Client and server-side scripts	Returns the internal ID of the sublist.
	Sublist.isChanged	boolean <code>true</code> <code>false</code> (read-only)	Client and server-side scripts	Indicates whether the sublist has changed on the record form.
	Sublist.isDisplay	boolean <code>true</code> <code>false</code> (read-only)	Client and server-side scripts	Indicates whether the sublist is displayed on the record form.
	Sublist.type	string (read-only)	Client and server-side scripts	Returns the sublist type.

N/record Module Script Samples

The following script samples demonstrate how to use the record module.

These samples use the `require` function, so that you can copy each script into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For more information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).



Important: Some of the values in these samples are placeholders. Before using these samples, replace all hardcoded 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.

The following example shows how to create and save a contact record.

```
/**
 *@NApiVersion 2.x
 */
require(['N/record'], function(record) {
    function createAndSaveContactRecord() {
        var nameData = {
            firstname: 'John',
            middlename: 'Doe',
            lastname: 'Smith'
        };
        var objRecord = record.create({
            type: record.Type.CONTACT,
            isDynamic: true
        });
    }
});
```

```

    });
    objRecord.setValue({
        fieldId: 'subsidiary',
        value: '1'
    });
    for ( var key in nameData) {
        if (nameData.hasOwnProperty(key)) {
            objRecord.setValue({
                fieldId: key,
                value: nameData[key]
            });
        }
    }
    var recordId = objRecord.save({
        enableSourcing: false,
        ignoreMandatoryFields: false
    });
}
createAndSaveContactRecord();
});

```

The following example shows how to create and save a contact record using Promise methods.

Note: To debug client scripts like the following, we recommend that you 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.

```

/**
 *@NApiVersion 2.x
 */
require(['N/record'], function(record) {
    function createAndSaveContactRecordWithPromise() {
        var nameData = {
            firstname: 'John',
            middlename: 'Doe',
            lastname: 'Smith'
        };
        var createRecordPromise = record.create.promise({
            type: record.Type.CONTACT,
            isDynamic: true
        });
        createRecordPromise.then(function(objRecord) {
            console.log('start evaluating promise content');
            objRecord.setValue({
                fieldId: 'subsidiary',
                value: '1'
            });
            for ( var key in nameData) {
                if (nameData.hasOwnProperty(key)) {
                    objRecord.setValue({
                        fieldId: key,
                        value: nameData[key]
                    });
                }
            }
        });
    }
}

```

```

    }
    var recordId = objRecord.save({
        enableSourcing: false,
        ignoreMandatoryFields: false
    });
}, function(e) {
    log.error('Unable to create contact', e.name);
});
}
createAndSaveContactRecordWithPromise();
});

```

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

```

/**
 *@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();
});

```

Note: For additional script samples that include subrecords, see [SuiteScript 2.0 Scripting Subrecords](#).

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

To debug client scripts like the following, we recommend that you 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.

```

/**
 * @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();
});

```

The following example 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 [Overview of Record Action and Macro APIs](#).

```

/**
 *@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
});
});

```

N/record Default Values

You can use SuiteScript 2.0 to specify record initialization parameters that default when creating, copying, loading, and transforming records. To enable this behavior, use the optional `defaultValues` parameter in the following APIs:

- `record.create(options)`
- `record.copy(options)`
- `record.transform(options)`

■ `record.load(options)`

The following table lists initialization types that are available to certain SuiteScript-supported records and the values they can contain.

Record	Initialization Type	Values
All SuiteScript-supported records that support form customization.	customform	<customformid>
Assembly Build	assemblyitem	<assemblyitemid>
Cash Refund	entity	<entityid>
Cash Sale	entity	<entityid>
Check	entity	<entityid>
Credit Memo	entity	<entityid>
Customer Payment	entity	<entityid>
Customer Refund	entity	<entityid>
Deposit	disablepaymentfilters	<disablepaymentfilters>
Estimate	entity	<entityid>
Expense Report	entity	<entityid>
Invoice	entity	<entityid>
Item Receipt	entity	<entityid>
Non-Inventory Part	subtype	sale resale purchase
Opportunity	entity	<entityid>
Other Charge Item	subtype	sale resale purchase
Purchase Order	entity	<entityid>
Return Authorization	entity	<entityid>
Sales Order	entity	<entityid>
Script Deployment	script	<scriptid>
Service	subtype	sale resale purchase
Tax Group	nexuscountry	<countrycode> See Country Codes Used for Initialization Parameters .
Tax Type	country	<countrycode> See Country Codes Used for Initialization Parameters .
Topic	parenttopic	<parenttopicid>
Vendor Bill	entity	<entityid>
Vendor Payment	entity	<entityid>
Work Order	assemblyitem	<assemblyitemid>

Country Codes Used for Initialization Parameters

If you are scripting the Tax Group or Tax Type records, you can initialize the record to source all values related to a specific country. In your script, use the country code for the *countrycodeid* value, for example:

```
record.create('taxgroup', {nexuscountry: 'AR'});
```

Country Code	Country Name
AD	Andorra
AE	United Arab Emirates
AF	Afghanistan
AG	Antigua and Barbuda
AI	Anguilla
AL	Albania
AM	Armenia
AO	Angola
AQ	Antarctica
AR	Argentina
AS	American Samoa
AT	Austria
AU	Australia
AW	Aruba
AX	Aland Islands
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BD	Bangladesh
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BH	Bahrain
BI	Burundi
BJ	Benin
BL	Saint Barthélemy
BM	Bermuda
BN	Brunei Darrussalam
BO	Bolivia

Country Code	Country Name
BQ	Bonaire, Saint Eustatius, and Saba
BR	Brazil
BS	Bahamas
BT	Bhutan
BV	Bouvet Island
BW	Botswana
BY	Belarus
BZ	Belize
CA	Canada
CC	Cocos (Keeling) Islands
CD	Congo, Democratic People's Republic
CF	Central African Republic
CG	Congo, Republic of
CH	Switzerland
CI	Cote d'Ivoire
CK	Cook Islands
CL	Chile
CM	Cameroon
CN	China
CO	Colombia
CR	Costa Rica
CU	Cuba
CV	Cape Verde
CW	Curacao
CX	Christmas Island
CY	Cyprus
CZ	Czech Republic
DE	Germany
DJ	Djibouti
DK	Denmark
DM	Dominica
DO	Dominican Republic
DZ	Algeria
EA	Ceuta and Melilla

Country Code	Country Name
EC	Ecuador
EE	Estonia
EG	Egypt
EH	Western Sahara
ER	Eritrea
ES	Spain
ET	Ethiopia
FI	Finland
FJ	Fiji
FK	Falkland Islands
FM	Micronesia, Federal State of
FO	Faroe Islands
FR	France
GA	Gabon
GB	United Kingdom
GD	Grenada
GE	Georgia
GF	French Guiana
GG	Guernsey
GH	Ghana
GI	Gibraltar
GL	Greenland
GM	Gambia
GN	Guinea
GP	Guadeloupe
GQ	Equatorial Guinea
GR	Greece
GS	South Georgia
GT	Guatemala
GU	Guam
GW	Guinea-Bissau
GY	Guyana
HK	Hong Kong
HM	Heard and McDonald Islands

Country Code	Country Name
HN	Honduras
HR	Croatia/Hrvatska
HT	Haiti
HU	Hungary
IC	Canary Islands
ID	Indonesia
IE	Ireland
IL	Israel
IM	Isle of Man
IN	India
IO	British Indian Ocean Territory
IQ	Iraq
IR	Iran (Islamic Republic of)
IS	Iceland
IT	Italy
JE	Jersey
JM	Jamaica
JO	Jordan
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KH	Cambodia
KI	Kiribati
KM	Comoros
KN	Saint Kitts and Nevis
KP	Korea, Democratic People's Republic
KR	Korea, Republic of
KW	Kuwait
KY	Cayman Islands
KZ	Kazakhstan
LA	Lao People's Democratic Republic
LB	Lebanon
LC	Saint Lucia
LI	Liechtenstein

Country Code	Country Name
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
LY	Libya
MA	Morocco
MC	Monaco
MD	Moldova, Republic of
ME	Montenegro
MF	Saint Martin
MG	Madagascar
MH	Marshall Islands
MK	Macedonia
ML	Mali
MM	Myanmar
MN	Mongolia
MO	Macau
MP	Northern Mariana Islands
MQ	Martinique
MR	Mauritania
MS	Montserrat
MT	Malta
MU	Mauritius
MV	Maldives
MW	Malawi
MX	Mexico
MY	Malaysia
MZ	Mozambique
NA	Namibia
NC	New Caledonia
NE	Niger
NF	Norfolk Island

Country Code	Country Name
NG	Nigeria
NI	Nicaragua
NL	Netherlands
NO	Norway
NP	Nepal
NR	Nauru
NU	Niue
NZ	New Zealand
OM	Oman
PA	Panama
PE	Peru
PF	French Polynesia
PG	Papua New Guinea
PH	Philippines
PK	Pakistan
PL	Poland
PM	St. Pierre and Miquelon
PN	Pitcairn Island
PR	Puerto Rico
PS	State of Palestine
PT	Portugal
PW	Palau
PY	Paraguay
QA	Qatar
RE	Reunion Island
RO	Romania
RS	Serbia
RU	Russian Federation
RW	Rwanda
SA	Saudi Arabia
SB	Solomon Islands
SC	Seychelles
SD	Sudan
SE	Sweden

Country Code	Country Name
SG	Singapore
SH	Saint Helena
SI	Slovenia
SJ	Svalbard and Jan Mayen Islands
SK	Slovak Republic
SL	Sierra Leone
SM	San Marino
SN	Senegal
SO	Somalia
SR	Suriname
SS	South Sudan
ST	Sao Tome and Principe
SV	El Salvador
SX	Sint Maarten
SY	Syrian Arab Republic
SZ	Swaziland
TC	Turks and Caicos Islands
TD	Chad
TF	French Southern Territories
TG	Togo
TH	Thailand
TJ	Tajikistan
TK	Tokelau
TM	Turkmenistan
TN	Tunisia
TO	Tonga
TP	East Timor
TR	Turkey
TT	Trinidad and Tobago
TV	Tuvalu
TW	Taiwan
TZ	Tanzania
UA	Ukraine
UG	Uganda

Country Code	Country Name
UM	US Minor Outlying Islands
US	United States
UY	Uruguay
UZ	Uzbekistan
VA	Holy See (City Vatican State)
VC	Saint Vincent and the Grenadines
VE	Venezuela
VG	Virgin Islands (British)
VI	Virgin Islands (USA)
VN	Vietnam
VU	Vanuatu
WF	Wallis and Futuna Islands
WS	Samoa
XK	Kosovo
YE	Yemen
YT	Mayotte
ZA	South Africa
ZM	Zambia
ZW	Zimbabwe

record.Column

Object Description	Encapsulates a column of a sublist on a standard or custom record. For a complete list of this object's properties, see Column Object Members . This object does not return a value, it returns information about the sublist column.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
```

```

    id: 275
  });

  var objSublist = objRecord.getSublist({
    sublistId: 'item'
  });

  var objColumn = objSublist.getColumn({
    fieldId: 'item'
  });

  if(objColumn.label === 'myLabel' ){
    //Perform an action
  }
  if(objColumn.type === 'checkbox'){
    //Perform an action
  }
  ...
  // Add additional code.

```

Column.id

Property Description	Returns the internal ID of the column. Note that the Column.id value is the same as the value that is passed into fieldID.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Column Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});

var objColumn = objSublist.getColumn({
  fieldId: 'item'
});

```

```
log.debug ({
  title: 'ID comparison',
  details: 'Note that objColumn.id = ' + objColumn.id + ' is the same as the value you passed in as fieldID.'
})...
// Add additional code.
```

Column.label

Property Description	Returns the internal ID of the column. This property does not return a value, it returns information about the column label.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Column Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});

var objColumn = objSublist.getColumn({
  fieldId: 'item'
});

if(objColumn.label === 'myLabel' ){
  //Perform an action
}
...
// Add additional code.
```

Column.sublistId

Property Description	Returns the internal ID of the standard or custom sublist that contains the column.
Type	string (read-only)

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Column Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

var objColumn = objSublist.getColumn({
    fieldId: 'item'
});
//Perform an action with the objColumn.sublistId value
...
// Add additional code.
```

Column.type

Property Description	Returns the column type. For more information on possible return values, see format.Type .
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Column Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
```

```

var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

var objColumn = objSublist.getColumn({
    fieldId: 'item'
});

if(objColumn.type === 'checkbox'){
    //Perform an action
}
...
// Add additional code.

```

record.Field

Object Description	<p>Encapsulates a body or sublist field on a standard or custom record. Use the following methods to access the Field object:</p> <ul style="list-style-type: none"> Record.getField(options) Record.getSublistField(options) Record.getCurrentSublistField(options) CurrentRecord.getField(options) CurrentRecord.getSublistField(options) <p>For a complete list of this object's methods and properties, see Field Object Members.</p>
Supported Script Types	<p>Client and server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/record Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

```

```

var objField = objSublist.getField({
    fieldId: 'item'
});
if(objField.label === 'myLabel' ){
    //Perform an action
}
if(objField.type === 'checkbox'){
    //Perform an action
}
...
// Add additional code.

```

Field.getSelectOptions(options)

Method Description	<p>Returns an array of available options on a standard or custom select, multi-select, or radio field as key-value pairs.</p> <div>  Important: You can only use this method on a record in dynamic mode. For additional information on dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>
Returns	<p>array</p> <p>Only the first 1,000 available options are returned in an array. If there are more than 1,000 available options, an empty array [] is returned. This function returns an array in the following format:</p> <pre>[{value: 5, text: 'abc'}, {value: 6, text: '123'}]</pre> <p>This function returns <code>Type Error</code> if the field is not a supported field for this method.</p>
Supported Script Types	<p>Client and server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Governance	None
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.filter	string	Required	<p>The search string to filter the select options that are returned.</p> <div>  Note: Filter values are case insensitive. </div>	2015.2
options.operator	string	Required	The following operators are supported:	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> contains (default) is startswith 	

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

var options = objField.getSelectOptions({
    filter : 'C',
    operator : 'startswith'
});

//Perform an action with the options array
...
// Add additional code.
```

Field.label

Property Description	Returns the UI label for a standard or custom field body or sublist field.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
```

```

    type: record.Type.SALES_ORDER,
    id: 275
  });

  var objSublist = objRecord.getSublist({
    sublistId: 'item'
  });

  var objField = objSublist.getField({
    fieldId: 'item'
  });

  if(objField.label === 'myLabel' ){
    //Perform an action
  }
  ...
  // Add additional code.

```

Field.id

Property Description	Returns the internal ID of a standard or custom body or sublist field.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});

var objField = objSublist.getField({
  fieldId: 'item'
});
//Perform an action with the objField.id value
...
// Add additional code.

```

Field.type

Property Description	Returns the type of a body or sublist field. For example, the value can return <code>text</code> , <code>date</code> , <code>currency</code> , <code>select</code> , <code>checkbox</code> , etc. For more information on possible return values, see format.Type .
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

var objField = objSublist.getField({
    fieldId: 'item'
});

if(objField.type === 'checkbox'){
    //Perform an action
}
...
// Add additional code.
```

Field.isMandatory

Property Description	Returns <code>true</code> if the standard or custom field is mandatory on the record form, or <code>false</code> otherwise.
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members

Since	2015.2
-------	--------

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});

var objField = objSublist.getField({
  fieldId: 'item'
});

if(objField.isMandatory){
  var options = {
    title: 'Incomplete Field:',
    message: 'Please complete this field.'
  };
  dialog.alert(options);
}
...
// Add additional code.
```

Field.sublistId

Property Description	Returns the sublist ID for the specified sublist field.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
```

```

    id: 275
  });

  var objSublist = objRecord.getSublist({
    sublistId: 'item'
  });

  var objField = objSublist.getField({
    fieldId: 'item'
  });

  //Perform an action with the objField.sublistId
  ...
  // Add additional code.

```

Field.isDisplay

Property Description	Returns <code>true</code> if the field is visible on the record form, or <code>false</code> if it is not.
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Field Object Members
Since	2015.2

record.Macro

Object Description	Encapsulates a record macro. For information about record macros, see Overview of Record Action and Macro APIs . Use the Record.getMacro(options) method to access the Macro object. For a complete list of this object's methods and properties, see Macro Object Members .
Supported Script Types	Client and server-side scripts. For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Methods and Properties	Macro Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code
...
var myMacro = record.getMacro({id: 'calculateTax'})
...

```

```
// Add additional code
```

Macro.execute(options)

Method Description	Performs a macro operation and returns its result in a plain JavaScript object. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	{notifications: [], response: {}}
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.params	Object	optional	The macro arguments.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
timesheet.executeMacro({id:'copyFromWeek', params: {weekOf : '7/10/2017', copyExact : true}});
...
// Add additional code
```

Macro.execute.promise(options)

Method Description	Performs a macro operation asynchronously. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Client Script Type .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.params	Object	optional	The macro arguments.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
myMacro.execute.promise().then(function(result){ /* do something with macro result */ });
...
// Add additional code
```

Macro(options)

Method Description	Performs a macro operation and returns its result in a plain JavaScript object. <div>  Note: Substitute Macro with the name of the macro you are executing. </div> For information about record macros, see Overview of Record Action and Macro APIs .
Returns	{notifications: [], response: {}}
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.params	Object	optional	The macro arguments.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
```

```

...
var calculateTax = recordObj.getMacro({id: 'calculateTax'});
calculateTax();
...
// Add additional code

```

Macro.promise(options)

Method Description	<p>Performs a macro operation asynchronously.</p> <div>  Note: Substitute Macro with the name of the macro you are executing. [link to conceptual topic] </div> <p>For information about record macros, see Overview of Record Action and Macro APIs.</p>
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Client Script Type .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.params	Object	optional	The macro arguments.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code
...
var calculateTax = recordObj.getMacro({id: 'calculateTax'});
calculateTax.promise();
...
// Add additional code

```

Macro.id

Property Description	The ID of the macro.
-----------------------------	----------------------

	For information about record macros, see Overview of Record Action and Macro APIs .
Type	string
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var id = macro.id; // get the id of the macro
...
// Add additional code
```

Macro.label

Property Description	The label of the macro. For information about record macros, see Overview of Record Action and Macro APIs .
Type	string
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var label = macro.label; // get the label of the macro
...
// Add additional code
```

Macro.description

Property Description	The description of the macro. For information about record macros, see Overview of Record Action and Macro APIs .
Type	string
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var description = macro.description; // get the description of the macro
...
// Add additional code
```

Macro.attributes

Property Description	The defined attributes of the macro. For information about record macros, see Overview of Record Action and Macro APIs .
Type	Object
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Parent Object	record.Macro
Sibling Object Members	Macro Object Members
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var attributes = macro.attributes; // get the attributes of the macro
```

```
...
// Add additional code
```

record.Record

Load the record module when you want to work with NetSuite records.

Object Description	<p>Encapsulates a NetSuite record.</p> <p>There are two modes you can operate in when you create, copy, load, or transform a record with SuiteScript 2.0: standard mode and dynamic mode.</p> <ul style="list-style-type: none"> ■ When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in standard mode, the record's body fields and sublist line items are not sourced, calculated, and validated until the record is saved (submitted) with Record.save(options). When you work with a record in standard mode, you do not need to set values in any particular order. After submitting the record, NetSuite processes the record's body fields and sublist line items in the correct order, regardless of the organization of your script. ■ When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in dynamic mode, the record's body fields and sublist line items are sourced, calculated, and validated in real-time. A record in dynamic mode emulates the behavior of a record in the UI. When you work with a record in dynamic mode, it is important that you set values in the same order you would within the UI. If you fail to do this, your results may not be accurate. <p>The record.create(options), record.copy(options), record.load(options), and record.transform(options) methods work in standard mode by default. If you want these methods to work in dynamic mode, you must pass in a specific argument. See the help topic for the applicable method for more information.</p> <p>For more information about standard and dynamic modes, see SuiteScript 2.0 – Standard and Dynamic Modes</p> <p>For a complete list of this object's methods and properties, see Record Object Members.</p>
Supported Script Types	<p>Client and server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/record Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: '6',
  isDynamic: true
});
...
// Add additional code.
```

Record.cancelLine(options)

Method Description	Cancels the currently selected line on a sublist.
---------------------------	---

	(dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	The <code>record.Record</code> object that called the method.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.cancelLine({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.commitLine(options)

Method Description	Commits the currently selected line on a sublist. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in standard mode, set a sublist field using Record.setSublistValue(options) .
Returns	The <code>record.Record</code> object that called the method.

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.commitLine({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.executeMacro(options)

Method Description	Performs macro operation and returns its result in a plain JavaScript object. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	An object with the macro results or <code>null</code> .
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .

Module	N/record Module
Sibling Object Members	Record Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The macro ID.
options.params	Object	optional	The macro arguments.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
if ('calculateTax' in macros) {
    macros.calculateTax();
}
...
// Add additional code
```

Record.executeMacro.promise(options)

Method Description	Performs macro operation and returns its result in a plain JavaScript object. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	Promise
Supported Script Types	Client-side scripts For additional information, see SuiteScript 2.0 Client Script Type .
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The macro ID.

Parameter	Type	Required / Optional	Description
options.params	Object	optional	The macro arguments.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
if ('calculateTax' in macros) {
    macros.calculateTax.promise();
}
...
// Add additional code
```

Record.findMatrixSublistLineWithValue(options)

Method Description	Returns the line number of the first instance where a specified value is found in a specified column of the matrix. Note that line and column indexing begins at 0 with SuiteScript 2.0. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2016.2
options.fieldId	string	required	The ID of the matrix field. See, How do I find a field's internal ID?	2016.2
options.value	number	required	The value to search for.	2016.2
options.column	number	required	The column number of the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2016.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var lineNumber = objRecord.findMatrixSublistLineWithValue({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.findSublistLineWithValue(options)

Method Description	Returns the line number for the first occurrence of a field value in a sublist. Note that line indexing begins at 0 with SuiteScript 2.0. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	A line number as a number, or -1 if not found.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2

Parameter	Type	Required / Optional	Description	Since
options.value	number Date string array boolean true false	optional	The value to search for.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or not defined.

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var lineNumber = objRecord.findSublistLineWithValue({
    sublistId: 'item',
    fieldId: 'item',
    value: 233
});
...
// Add additional code.
```

Record.getCurrentMatrixSublistValue(options)

Method Description	Gets the value for the currently selected line in the matrix. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number Date string array boolean true false
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the matrix field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var matrixValue = objRecord.getCurrentMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
// Add additional code.
```

Record.getCurrentSublistField(options)

Method Description	Returns metadata about a sublist field. (dynamic mode only— see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Field
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistFieldMetadata = objRecord.getCurrentSublistField({
    sublistId: 'item',
    fieldId: 'item',
});
...
// Add additional code.
```

Record.getCurrentSublistIndex(options)

Method Description	Returns the line number of the currently selected line. Note that line indexing begins at 0 with SuiteScript 2.0. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist.	2015.2

Parameter	Type	Required / Optional	Description	Since
			This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var currIndex = objRecord.getCurrentSublistIndex({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.getCurrentSublistSubrecord(options)

Method Description	Gets the subrecord for the associated sublist field on the current line. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objSubrecord = objRecord.getCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item'
});
...
// Add additional code.
```

Record.getCurrentSublistText(options)

Method Description	Returns a text representation of the field value in the currently selected line. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	string <div>  Note: For multiselect fields, returns an array. </div>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var fieldName = objRecord.getCurrentSublistText({
    sublistId: 'item',
    fieldId: 'item'
});
...
// Add additional code.
```

Record.getCurrentSublistValue(options)

Method Description	Returns the value of a sublist field on the currently selected sublist line. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number Date string array boolean true false
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field.	2015.2

Parameter	Type	Required / Optional	Description	Since
			See, How do I find a field's internal ID?	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistValue = objRecord.getCurrentSublistValue({
    sublistId: 'item',
    fieldId: 'item'
});
...
// Add additional code.
```

Record.getField(options)

Method Description	Returns a field object from a record. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Field
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field.	2015.2

Parameter	Type	Required / Optional	Description	Since
			See, How do I find a field's internal ID?	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objField = objRecord.getField({
    fieldId: 'item'
});
...
// Add additional code.
```

Record.getFields()

Method Description	Returns the body field names (internal ids) of all the fields in the record, including machine header field and matrix header fields. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	string[]
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objFields = objRecord.getFields();
...
```



```
// Add additional code.
```

Record.getLineCount(options)

Method Description	Returns the number of lines in a sublist. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var numLines = objRecord.getLineCount({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.getMacro(options)

Method Description	Provides a macro to be executed. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	Function to be executed for the macro.

Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The macro ID.

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required parameter is missing.
SSS_INVALID_MACRO_ID	A macro does not exist on the record.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var macro = recordObj.getMacro({id: 'calculateTax'});
...
// Add additional code
```

Record.getMacros(options)

Method Description	Provides a plain JavaScript object of available macro objects defined for a record type, indexed by the Macro ID. The object returns one or more record.Macro objects. If there are no macros available for the specified record type, an empty object is returned. For information about record macros, see Overview of Record Action and Macro APIs .
Returns	Object
Supported Script Types	Client and server scripts For additional information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Record Object Members

Since	2018.2
-------	--------

Errors

Error Code	Thrown If
SSS_INVALID_RECORD_TYPE	The specified record type is invalid.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code
...
var macroList = recordObj.getMacros();
...
// Add additional code
```

Record.getMatrixHeaderCount(options)

Method Description	Returns the number of columns for the specified matrix. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
....
var numLines = objRecord.getMatrixHeaderCount({
    sublistId: 'item',
    fieldId: 'item'
});
...
// Add additional code.
```

Record.getMatrixHeaderField(options)

Method Description	Gets the field for the specified header in the matrix. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Field
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objField = objRecord.getMatrixHeaderField({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
// Add additional code.
```

Record.getMatrixHeaderValue(options)

Method Description	Gets the value for the associated header in the matrix. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number Date string array boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var value = objRecord.getMatrixHeaderValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12
});
...
// Add additional code.
```

Record.getMatrixSublistField(options)

Method Description	Gets the field for the specified sublist in the matrix. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Field
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objField = objRecord.getMatrixSublistField({
    sublistId: 'item',
    fieldId: 'item',
    column: 12,
    line: 3
});
...
// Add additional code.
```

Record.getMatrixSublistValue(options)

Method Description	Gets the value for the associated field in the matrix. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number Date string array boolean true false
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
options.column	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var value = objRecord.getMatrixSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    column: 12,
    line: 3
});
...
// Add additional code.
```

Record.getSublist(options)

Method Description	Returns the specified sublist. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Sublist
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist.	2015.2

Parameter	Type	Required / Optional	Description	Since
			This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objSublist = objRecord.getSublist({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.getSublists()

Method Description	Returns all the names of all the sublists. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	string[]
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistName = objRecord.getSublists();
...
// Add additional code.
```

Record.getSublistField(options)

Method Description	Returns a field object from a sublist. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Field

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objField = objRecord.getSublistField({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
// Add additional code.
```

Record.getSublistFields(options)

Method Description	Returns all the field names in a sublist. (standard and dynamic mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
---------------------------	--

Returns	string[]
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var field = objRecord.getSublistFields({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.getSublistSubrecord(options)

Method Description	Gets the subrecord associated with a sublist field. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in dynamic mode, get a sublist subrecord using the following methods: <ol style="list-style-type: none"> 1. Record.selectLine(options) 2. Record.hasCurrentSublistSubrecord(options) 3. Record.getCurrentSublistSubrecord(options)
Returns	record.Record

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
options.line	number	required	The line number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objSubRecord = objRecord.getSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
// Add additional code.
```

Record.getSublistText(options)

Method Description	Returns the value of a sublist field in a text representation. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	string  Note: For multiselect fields, returns an array.
Supported Script Types	Client and server-side scripts. For more information, see SuiteScript 2.0 Script Types .

	Limitations exist on how this method can be used in standard (deferredDynamic) mode. For details, refer to the description of the SSS_INVALID_API_USAGE error code in the Errors table. In dynamic mode, you can use <code>getSublistText()</code> without limitation.
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_INVALID_API_USAGE	<p>Invoked in certain cases when deferredDynamic mode is being used. For example, if <code>Record.isDynamic</code> is set to false, this error can be invoked in both of the following situations:</p> <ul style="list-style-type: none"> ■ If the record object was created by <code>record.copy()</code>, <code>record.create()</code>, or <code>record.transform()</code>, and the script attempts to use <code>getSublistText()</code> without first using <code>setSublistText()</code> for the same field. ■ If the record object was created by <code>record.load()</code>, and the script uses <code>setSublistValue()</code> on a field before using <code>getSublistText()</code> for the same field. <p>This guidance also affects user event scripts that instantiate records by using the <code>newRecord</code> or <code>oldRecord</code> object provided by the script context. These records always use deferredDynamic mode. For that reason, this error appears in both of the following situations:</p> <ul style="list-style-type: none"> ■ When a user event script executes on a record that is being newly created, and the script attempts to use <code>getSublistText()</code> without first using <code>setSublistText()</code> for the same field. ■ When a user event script executes on an existing record, and the script uses <code>setSublistValue()</code> on a field before using <code>getSublistText()</code> for the same field. <p>For more information, see SuiteScript 2.0 – Standard and Dynamic Modes.</p>
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Error Code	Thrown If
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistFieldName = objRecord.getSublistText({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
// Add additional code.
```

Record.getSublistValue(options)

Method Description	Returns the value of a sublist field. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	number Date string array boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_INVALID_API_USAGE	Invoked prior to using setSublistValue in standard record mode.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistFieldValue = objRecord.getSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
// Add additional code.
```

Record.getSubrecord(options)

Method Description	Gets the subrecord for the associated field. This method is not available for subrecords. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field.	2015.2

Parameter	Type	Required / Optional	Description	Since
			See, How do I find a field's internal ID?	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
FIELD_1_IS_NOT_A_SUBRECORD_FIELD	The specified field is not a subrecord field.
FIELD_1_IS_DISABLED_YOU_CANNOT_APPLY_SUBRECORD_OPERATION_ON_THIS_FIELD	The specified field is disabled.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var sublistFieldValue = objRecord.getSubrecord({
    fieldId: 'idnumber'
});
...
// Add additional code.
```

Record.getText(options)

Method Description	Returns the text representation of a field value. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	string <div>  Note: For multiselect fields, returns an array. </div>
Supported Script Types	Client and server-side scripts. For more information, see SuiteScript 2.0 Script Types . In dynamic mode, you can use <code>getText()</code> without limitation but, in standard mode, limitations exist. In standard mode, you can use this method only in the following cases: <ul style="list-style-type: none"> You can use <code>getText()</code> on any field where the script has already used <code>setText()</code>. If you are loading or copying a record, you can use <code>getText</code> on any field except those where the script has already changed the value by using <code>setValue()</code>. For more details, refer to the description of the <code>SSS_INVALID_API_USAGE</code> error code in the Errors table.
Governance	None
Module	N/record Module

Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See, How do I find a field's internal ID?	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_API_USAGE	<p>Invoked in certain cases when standard mode is being used. For example, if Record.isDynamic is set to false, the SSS_INVALID_API_USAGE error can be invoked in the following situations:</p> <ul style="list-style-type: none"> ■ If the record object was created by record.create() or record.transform(), and the script attempts to use getText() without first using setText() for the same field. ■ The record object was created by record.copy() or record.load(), and the script uses setValue() on a field before using getText() for the same field. <p>Similar guidance affects user event scripts that instantiate records by using the newRecord or oldRecord object provided by the script context. In these cases, standard mode is always used. For that reason, the SSS_INVALID_API_USAGE error appears when a user event executes on one of these objects in the following situations:</p> <ul style="list-style-type: none"> ■ When the script executes on a record that is being created, and the script attempts to use getText() without first using setText() for the same field. ■ When the script executes on an existing record or on a record being created through copying, and the script uses setValue() on a field before using getText() for the same field.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Syntax Sample 1
// Add additional code.
...
var fieldidname = objRecord.getText({
    fieldId: 'item'
});
...
```

```
// Add additional code.

// Syntax Sample 2
// Add additional code.
...
myString = 'Date is: ' + record.getText({fieldId: 'datechanged'});
// "Date is: 3/27/2017 9:55:38am"
...
// Add additional code
```

Record.getValue(options)

Method Description	Returns the value of a field. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	<p>number Date string array boolean <code>true</code> <code>false</code></p> <p>Returns a JavaScript Date object for date/time field queries. To return a string for date/time field queries, use Record.getText(options). Date/time fields: DATE, DATETIME, DATETIMETZ, TIMEOFDAY.</p> <div>  Note: If the returned date object is implicitly converted to a string, the value is converted using the browser's setting for time zone. </div> <p>Checkbox fields return values of T or F. If you include checkbox field return values in scripts, be sure to use T and F, instead of boolean values, <code>true</code> and <code>false</code>.</p>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field. See, How do I find a field's internal ID?	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_API_USAGE	Invoked in standard mode, if you use <code>setText</code> on a field and then use <code>getValue</code> on the same field.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var value = objRecord.getValue({
    fieldId: 'item'
});
...
// Add additional code.
```

Record.hasCurrentSublistSubrecord(options)

Method Description	Returns a value indicating whether the associated sublist field has a subrecord on the current line. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a subrecord. See, How do I find a field's internal ID?	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var hasSubrecord = objRecord.hasCurrentSublistSubrecord({
    sublistId: 'item',
```

```

        fieldId: 'item'
    });
    ...
    // Add additional code.

```

Record.hasSublistSubrecord(options)

Method Description	Returns a value indicating whether the associated sublist field contains a subrecord. This method is not available for subrecords. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a subrecord. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var hasSubrecord = objRecord.hasSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item',
    line: 3
});
...
// Add additional code.

```

Record.hasSubrecord(options)

Method Description	Returns a value indicating whether the field contains a subrecord. This method is not available for subrecords. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of the field that may contain a subrecord. See, How do I find a field's internal ID?	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var hasSubrecord = objRecord.hasSubrecord({
    fieldId: 'item'
});
...
// Add additional code.
```

Record.insertLine(options)

Method Description	Inserts a sublist line. When you insert a line with this method, all succeeding lines are moved and the total line count is increased. Essentially, succeeding lines are committed to a new sublist line with a new line number. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.line	number	required	The line number to insert. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2
options.ignoreRecalc	boolean true false	optional	If set to true, scripting recalculation is ignored. The default value is false.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example that uses insertLine(), see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.insertLine({
    sublistId: 'attendee',
    line: 2,
});
objRecord.setCurrentSublistValue({
    sublistId: 'attendee',
    fieldId: 'attendee',
    value: 838
});
objRecord.commitLine({
    sublistId: 'attendee'
});
...
```

```
// Add additional code.
```

For script examples that use other N/record methods, see [N/record Module Script Samples](#).

Record.removeCurrentSublistSubrecord(options)

Method Description	Removes the subrecord for the associated sublist field on the current line. This method is not available for subrecords. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.removeCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'item'
});
...
// Add additional code.
```

Record.removeLine(options)

Method Description	Removes a sublist line.
---------------------------	-------------------------

	(dynamic and standard mode — see SuiteScript 2.0 - Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.line</code>	number	required	The line number of the sublist to remove. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2
<code>options.ignoreRecalc</code>	boolean true false	optional	If set to true, scripting recalculation is ignored. The default value is false.	2015.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.
<code>SSS_INVALID_SUBLIST_OPERATION</code>	A required argument is invalid or the sublist is not editable.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.removeLine({
    sublistId: 'item',
    line: 3,
    ignoreRecalc: true
});
...
// Add additional code.
```


Record.removeSublistSubrecord(options)

Method Description	Removes the subrecord for the associated sublist field. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in dynamic mode, remove a sublist subrecord using the following methods: <ol style="list-style-type: none"> 1. Record.selectLine(options) 2. Record.hasCurrentSublistSubrecord(options) 3. Record.removeCurrentSublistSubrecord(options) 4. Record.commitLine(options)
Returns	<code>record.Record</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number in the sublist that contains the subrecord to remove. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.removeSublistSubrecord({
  sublistId: 'item',
  fieldid: 'item',
  line: 3
});
...
// Add additional code.
```

Record.removeSubrecord(options)

Method Description	Removes the subrecord for the associated field. This method is not available for subrecords. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field. See, How do I find a field's internal ID?	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.removeSubrecord({
    fieldid: 'item'
});
...
// Add additional code.
```

Record.save(options)

Method Description	Submits a new record or saves edits to an existing record. When working with records in standard mode, you must submit and then load the record to obtain sourced, validated, and calculated field values. This method is not available to subrecords. <div> Note: This method has an asynchronous counterpart you can use with client scripts. See Record.save.promise(options).</div>
Returns	A number representing the internal ID of the new or updated record.

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 20 usage units Custom records: 4 usage units All other records: 10 usage units
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.enableSourcing</code>	boolean true false	optional	<p>Enables sourcing during the record update. If set to <code>true</code>, sources dependent field information for empty fields. Defaults to <code>false</code> – dependent field values are not sourced.</p> <div>  Important: This parameter applies to records in standard mode only. When working with records in dynamic mode, field values are always sourced and the value you provide for <code>enableSourcing</code> is ignored. See SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.ignoreMandatoryFields</code>	boolean true false	optional	<p>Disables mandatory field validation for this save operation. If set to <code>true</code>, all standard and custom fields that were made mandatory through customization are ignored. All fields that were made mandatory through company preferences are also ignored. By default, this parameter is <code>false</code>.</p> <div>  Important: Use the <code>ignoreMandatoryFields</code> argument with caution. This argument should be used mostly with Scheduled scripts, rather than User Event scripts. This ensures that UI users do not bypass the business logic enforced through form customization. </div>	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var recordId = objRecord.save({
```

```

    enableSourcing: true,
    ignoreMandatoryFields: true
  });
  ...
  // Add additional code.

```

Record.save.promise(options)

Method Description	Submits a new record asynchronously or saves edits to an existing record asynchronously. This method is not available to subrecords. <div>  Note: For information about the parameters and errors thrown for this method, see Record.save(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 20 usage units Custom records: 4 usage units All other records: 10 usage units
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.enableSourcing</code>	boolean true false	optional	Enables sourcing during the record update. If set to <code>true</code> , sources dependent field information for empty fields. Defaults to <code>false</code> – dependent field values are not sourced. <div>  Important: This parameter applies to records in standard mode only. When working with records in dynamic mode, field values are always sourced and the value you provide for <code>enableSourcing</code> is ignored. See SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.ignoreMandatoryFields</code>	boolean true false	optional	Disables mandatory field validation for this save operation. If set to <code>true</code> , all standard and custom fields that were made mandatory through customization are ignored. All fields that were made mandatory through company preferences are also ignored. By default, this parameter is <code>false</code> .	2015.2

Parameter	Type	Required / Optional	Description	Since
			 Important: Use the <code>ignoreMandatoryFields</code> argument with caution. This argument should be used mostly with Scheduled scripts, rather than User Event scripts. This ensures that UI users do not bypass the business logic enforced through form customization.	

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.
...
var recordId = objRecord.save.promise({
    enableSourcing: true,
    ignoreMandatoryFields: true
});
...
// Add additional code.
```

Record.selectLine(options)

Method Description	Selects an existing line in a sublist. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in standard mode, set a sublist field using Record.setSublistValue(options) .
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.line</code>	number	required	The line number to select in the sublist. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var lineNum = objRecord.selectLine({
    sublistId: 'item',
    line: 3
});
...
// Add additional code.
```

Record.selectNewLine(options)

Method Description	Selects a new line at the end of a sublist. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var lineNum = objRecord.selectNewLine({
    sublistId: 'item'
});
...
// Add additional code.
```

Record.setCurrentMatrixSublistValue(options)

Method Description	Sets the value for the line currently selected in the matrix. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2015.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The options.value type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setCurrentMatrixSublistValue({
  sublistId: 'item',
  fieldId: 'item',
  column: 3,
  value: false,
  ignoreFieldChange: true,
  fireSlavingSync: true
});
...
// Add additional code.
```

Record.setCurrentSublistText(options)

Method Description	Sets the value for the field in the currently selected line by a text representation. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	<code>record.Record</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None

Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
options.text	string	required	The text to set the value to.	2015.2
options.ignoreFieldChange	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
A_SCRIPT_IS_ATTEMPTING_TO_EDIT_THE_1_SUBLIST_THIS_SUBLIST_IS_CURRENTLY_IN_READONLY_MODE_AND_CANNOT_BE_EDITED_CALL	A read-only field is being edited.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setCurrentSublistText({
    sublistId: 'item',
    fieldId: 'item',
    text: 'value',
    ignoreFieldChange: true
});
...
// Add additional code.
```

Record.setCurrentSublistValue(options)

Method Description	Sets the value for the field in the currently selected line. (dynamic mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in standard mode, set a sublist field using Record.setSublistValue(options) .
Returns	<code>record.Record</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2015.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
<code>INVALID_FLD_VALUE</code>	The <code>options.value</code> type does not match the field type.

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
A_SCRIPT_IS_ATTEMPTING_TO_EDIT_THE_1_SUBLIST_THIS_SUBLIST_IS_CURRENTLY_IN_READ_ONLY_MODE_AND_CANNOT_BE_EDITED_CALL_	ONLY MODE AND a read-only sublist field.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setCurrentSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    value: true,
    ignoreFieldChange: true
});
...
// Add additional code.
```

Record.setMatrixHeaderValue(options)

Method Description	Sets the value for the associated header in the matrix. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
<code>options.column</code>	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2015.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The <code>options.value</code> type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setMatrixHeaderValue({
  sublistId: 'item',
  fieldId: 'item',
  column: 3,
  value: false,
  ignoreFieldChange: true,
  fireSlavingSync: true
});
...
// Add additional code.
```

Record.setMatrixSublistValue(options)

Method Description	Sets the value for the associated field in the matrix. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	<code>record.Record</code>

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublistId	string	required	The internal ID of the sublist that contains the matrix. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
options.fieldId	string	required	The internal ID of the matrix field. See, How do I find a field's internal ID?	2015.2
options.column	number	required	The column number for the field. Note that column indexing begins at 0 with SuiteScript 2.0.	2015.2
options.line	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2
options.value	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2015.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The options.value type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setMatrixSublistValue({
  sublistId: 'item',
  fieldId: 'item',
```

```

column: 12,
line: 3,
value: true
});
...
// Add additional code.

```

Record.setSublistText(options)

Method Description	Sets the value of a sublist field by a text representation. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in dynamic mode, set a sublist field text using the following methods: <ol style="list-style-type: none"> 1. Record.selectLine(options) 2. Record.setCurrentSublistText(options) 3. Record.commitLine(options)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number for the field. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2
<code>options.text</code>	string	required	The text to set the value to.	2015.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.
<code>SSS_INVALID_SUBLIST_OPERATION</code>	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setSublistText({
    sublistId: 'item',
    fieldId: 'item',
    line: 3,
    text: 'value'
});
...
// Add additional code.
```

Record.setSublistValue(options)

Method Description	Sets the value of a sublist field. (standard mode only — see SuiteScript 2.0 – Standard and Dynamic Modes) When working in dynamic mode, set a sublist field value using the following methods: <ol style="list-style-type: none"> 1. Record.selectLine(options) 2. Record.setCurrentSublistValue(options) 3. Record.commitLine(options)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.sublistId</code>	string	required	The internal ID of the sublist. This value is displayed in the Records Browser. For more information, see the help topic Using the SuiteScript Records Browser .	2015.2
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom sublist field. See, How do I find a field's internal ID?	2015.2
<code>options.line</code>	number	required	The line number of the sublist. Note that line indexing begins at 0 with SuiteScript 2.0.	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.value</code>	number Date string array boolean <code>true</code> <code>false</code>	required	<p>The value to set the sublist field to. The value type must correspond to the field type being set. For example:</p> <ul style="list-style-type: none"> Text, Radio and Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	2015.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The options.value type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.
SSS_INVALID_SUBLIST_OPERATION	A required argument is invalid or the sublist is not editable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    line: 3,
    value: true
});
...
// Add additional code.
```

Record.setText(options)

Method Description	Sets the value of the field by a text representation. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of a standard or custom body field. See, How do I find a field's internal ID?	2015.2
options.text	string array	required	The text or texts to change the field value to. <ul style="list-style-type: none">■ If the field type is multiselect:<ul style="list-style-type: none">□ This parameter accepts an array of string values.□ This parameter accepts a null value. Passing in null deselects all currently selected values.■ If the field type is not multiselect, this parameter accepts only a single string value.	2015.2
options.ignoreFieldChange	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
objRecord.setText({
  fieldId: 'item',
  text: 'value',
  ignoreFieldChange: true
});
...
// Add additional code.
```

Record.setValue(options)

Method Description	Sets the value of a field.
--------------------	----------------------------

	(dynamic and standard mode — see SuiteScript 2.0 - Standard and Dynamic Modes)
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.fieldId</code>	string	required	The internal ID of a standard or custom body field. See, How do I find a field's internal ID?	2015.2
<code>options.value</code>	number Date string array boolean true false	required	The value to set the field to. The value type must correspond to the field type being set. For example: <ul style="list-style-type: none">Text, Radio, Select and Multi-Select fields accept string values.Checkbox fields accept Boolean values.Date and DateTime fields accept Date values.Integer, Float, Currency and Percent fields accept number values.	2015.2
<code>options.ignoreFieldChange</code>	boolean true false	optional	If set to <code>true</code> , the field change and slaving event is ignored. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
INVALID_FLD_VALUE	The options.value type does not match the field type.
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.  
...
```

```
objRecord.setValue({
    fieldId: 'item',
    value: true,
    ignoreFieldChange: true
});
...
// Add additional code.
```

Record.id

Property Description	The internal ID of a specific record. This property is not available to subrecords. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Record.isDynamic

Property Description	<p>Indicates whether the record is in dynamic or standard mode.</p> <ul style="list-style-type: none"> ■ If set to <code>true</code>, the record is currently in dynamic mode. If set to <code>false</code>, the record is currently in standard mode. <ul style="list-style-type: none"> □ When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in standard mode, the record's body fields and sublist line items are not sourced, calculated, and validated until the record is saved (submitted) with Record.save(options). When you work with a record in standard mode, you do not need to set values in any particular order. After submitting the record, NetSuite processes the record's body fields and sublist line items in the correct order, regardless of the organization of your script. □ When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in dynamic mode, the record's body fields and sublist line items are sourced, calculated, and validated in real-time. A record in dynamic mode emulates the behavior of a record in the UI. When you work with a record in dynamic mode, it is important that you set values in the same order you would within the UI. If you fail to do this, your results may not be accurate. <p>This value is set when the record is created or accessed. (dynamic and standard mode — see SuiteScript 2.0 – Standard and Dynamic Modes)</p>
Type	boolean <code>true</code> <code>false</code> (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
if (record.isDynamic) {
  ...
}
...
// Add additional code.
```

Record.type

Property Description	<p>The record type. Note the following:</p> <ul style="list-style-type: none">■ When working with an instance of a standard NetSuite record type, set this value by using the record.Type enum.■ When working with an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. <p>This property is not available to subrecords. (dynamic and standard modes — see SuiteScript 2.0 – Standard and Dynamic Modes)</p>
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Record Object Members
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...

// Start the process of creating an employee record.

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

// Start the process of creating an instance of a custom record type.

var customRecord = record.create({
  type: 'customrecord_book',
```

```

        isDynamic: true
    });

    ...
    // Add additional code.

```

Note: Supported standard record types are described in the [SuiteScript Records Browser](#). Refer also to [SuiteScript Supported Records](#). For help working with custom record types, see the help topic [Custom Record](#).

record.Sublist

Object Description	Encapsulates a sublist on a standard or custom record. For a complete list of this object's methods and properties, see Sublist Object Members .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});
if(objSublist.type === 'inlineeditor'){
    //Perform an action
}
...
// Add additional code.

```

Sublist.getColumn(options)

Method Description	Returns a column in the sublist.
Returns	record.Column
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None

Module	N/record Module
Sibling Object Members	Sublist Object Members
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.fieldId	string	required	The internal ID of the column field in the sublist. See, How do I find a field's internal ID?	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

Example 1

```
// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

var objColumn = objSublist.getColumn({
    fieldId: 'item'
});

if(objColumn.type === 'checkbox'){
    //Perform an action
}
...
// Add additional code.
```

Example 2

This example loops through each line of the items sublist on a sales order record.

```
// Add additional code
...
onRequest: function(context) {
    var recordObj = record.create({type: record.Type.SALES_ORDER});
    var columnList = recordObj.getSublistFields({sublistId: 'item'});
    var sublistObj = recordObj.getSublist({sublistId: 'item'});
```

```

    for (var i = 0; i < columnList.length; i++) {
        var columnId = columnList[i];
        var columnObj = sublistObj.getColumn({fieldId: columnId});
        if (columnObj !== null) {
            log.debug('[Column id] = ' + columnObj.id + ' [Column type] = ' + columnObj.type +
                ' [Column label] = ' + columnObj.label);
        }
    }
}
...
// Add additional code

```

Sublist.id

Property Description	Returns the internal ID of the sublist.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Sublist Object Members
Since	2015.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});
//Perform an action with the objSublist.id value
...
// Add additional code.

```

Sublist.isChanged

Property Description	Indicates whether the sublist has changed on the record form.
Type	boolean <code>true</code> <code>false</code> (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module

Sibling Object Members	Sublist Object Members
Since	2015.2

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});

if(objSublist.isChanged){
  //Perform an action when objSublist.isChanged is true
}
...
// Add additional code.
```

Sublist.isDisplay

Property Description	Indicates whether the sublist is displayed on the record form.
Type	boolean true false (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Sublist Object Members
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 275
});

var objSublist = objRecord.getSublist({
  sublistId: 'item'
});
```



```

if(objSublist.isDisplay){
    //Perform an action when objSublist.isDisplay is true
}
...
// Add additional code.

```

Sublist.type

Property Description	Returns the sublist type. For more information on sublist types, see serverWidget.SublistType . <div>  Important: Sublist.type will return a lower case string representing the sublist type. For example, <code>inlineeditor</code> not <code>INLINEEDITOR</code>. </div>
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Sibling Object Members	Sublist Object Members
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```

// Add additional code.
...
var objRecord = record.load({
    type: record.Type.SALES_ORDER,
    id: 275
});

var objSublist = objRecord.getSublist({
    sublistId: 'item'
});

if(objSublist.type === 'inlineeditor'){
    //Perform an action
}
...
// Add additional code.

```

record.attach(options)

Method Description	Attaches a record to another record.
---------------------------	--------------------------------------

	 Note: For the promise version of this method, see <code>record.attach.promise(options)</code> . Note that promises are only supported in client scripts.
Returns	void
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.record</code>	record.Record	required	The record to attach.	2015.2
<code>options.record.type</code>	string	required	The type of record to attach. Set this value using the record.Type enum. To attach a file from the file cabinet to a record, set type to <code>file</code> .	2015.2
<code>options.record.id</code>	number string	required	The internal ID of the record to attach.	2015.2
<code>options.to</code>	record.Record	required	The record that the <code>options.record</code> gets attached to.	2015.2
<code>options.to.type</code>	string	required	The record type of the record to attach to. Set the value using the record.Type enum. To attach a file from the file cabinet to a record, set type to <code>file</code> .	2015.2
<code>options.to.id</code>	number string	required	The internal ID of the record to attach to.	2015.2
<code>options.attributes</code>	Object	optional	The name-value pairs containing attributes for the attachment. By default, this value is null.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/record Module Script Samples .
--

```
// Add additional code.
```

```

...
var id = record.attach({
  record: {
    type: 'file',
    id: '200'
  },
  to: {
    type: 'customer',
    id: '90'
  }
});
...
// Add additional code.

```

record.attach.promise(options)

Method Description	Attaches a record asynchronously to another record. <div>  Note: For information about the parameters and errors thrown for this method, see record.attach(options). For more information about promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/record Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.record	record.Record	required	The record to attach.	2015.2
options.record.type	string	required	The type of record to attach. Set the value using the record.Type enum. To attach a file from the file cabinet to a record, set type to <code>file</code> .	2015.2
options.record.id	number string	required	The internal ID of the record to attach.	2015.2
options.to	record.Record	required	The record that the <code>options.record</code> gets attached to.	2015.2
options.to.type	string	required	The record type of the record to attach to. Set the value using the record.Type enum. To attach a file from the file cabinet to a record, set type to <code>file</code> .	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.to.id</code>	number string	required	The internal ID of the record to attach to.	2015.2
<code>options.attributes</code>	Object	optional	The name-value pairs containing attributes for the attachment. By default, this value is null.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.
...
function attachRecord() {

    var attachRecordPromise = record.attach.promise({
        record: {
            type: record.Type.CONTACT,
            id: '97'
        },
        to: {
            type: record.Type.OPPORTUNITY,
            id: '16'
        }
    });

    attachRecordPromise.then(function() {

        // Add any other needed logic that shouldn't execute until
        // after the contact record is attached to the opportunity.

        log.debug({
            title: 'Record updated',
            details: 'Attachment successful'
        });

    }, function(e) {
        log.error({
            title: e.name,
            details: e.message
        });
    });
}
...
// Add additional code.
```

record.copy(options)

Method Description	Creates a new record by copying an existing record in NetSuite. <div>  Note: For the promise version of this method, see record.copy.promise(options). Note that promises are only supported in client scripts. </div>
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	required	The record type. Use the following guidelines: <ul style="list-style-type: none"> When copying an instance of a standard NetSuite record type, set this value by using the record.Type enum. When copying an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
<code>options.id</code>	number	required	The internal ID of the existing record instance in NetSuite.	2015.2
<code>options.isDynamic</code>	boolean <code>true</code> <code>false</code>	optional	Determines whether the new record is created in dynamic mode. <ul style="list-style-type: none"> If set to <code>true</code>, the new record is created in dynamic mode. If set to <code>false</code>, the new record is created in standard mode. By default, this value is <code>false</code> . <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	Name-value pairs containing default values of fields in the new record. By default, this value is null. For a list of available record default values, see N/record Default Values in the NetSuite Help Center.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.copy({
  type: record.Type.SALES_ORDER,
  id: 157,
  isDynamic: true,
  defaultValues: {
    entity: 107
  }
});
...
// Add additional code.
```

record.copy.promise(options)

Method Description	Creates a new record asynchronously by copying an existing record in NetSuite. <div> Note: For information about the parameters and errors thrown for this method, see record.copy(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The record type. Use the following guidelines:	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> When copying an instance of a standard NetSuite record type, set this value by using the record.Type enum. When copying an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	
<code>options.id</code>	number	required	The internal ID of the existing record instance in NetSuite.	2015.2
<code>options.isDynamic</code>	boolean true false	optional	<p>Determines whether the new record is created in dynamic mode.</p> <ul style="list-style-type: none"> If set to <code>true</code>, the new record is created in dynamic mode. If set to <code>false</code>, the new record is created in standard mode. <p>By default, this value is <code>false</code>.</p> <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	Name-value pairs containing default values of fields in the new record. By default, this value is null.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.
...
function copyRecord() {

    // Copy an instance of a standard record type.

    var copyRecordPromise = record.copy.promise({
        type: record.Type.PHONE_CALL,
        id: 165
    });

    // Note: To copy an instance of a custom record type,
    // use the record type's string ID instead of the record
    // module's Type enum. For example:
    // type: 'customrecord_feature'
```

```

copyRecordPromise.then(function(recordObject) {

    recordObject.setValue({
        fieldId: 'title',
        value: 'Sprint 5 bug triage'
    });

    recordObject.setValue({
        fieldId: 'message',
        value: 'Please review the PowerPoint prior to the call.'
    });

    var recordId = recordObject.save();

    // Add any other needed logic that shouldn't execute until
    // after the record is copied.

    log.debug({
        title: 'Record saved',
        details: 'Id of new record: ' + recordId
    });

    }, function(e) {
        log.error({
            title: e.name,
            details: e.message
        });
    });
}
...
// Add additional code.

```

record.create(options)

Method Description	Creates a new record. <div>  Note: For the promise version of this method, see record.create.promise(options). Note that promises are only supported in client scripts. </div>
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript Object.

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	required	<p>The record type. This value determines the Record.type property of the record that is created. This property is read-only on an existing record. Use the following guidelines:</p> <ul style="list-style-type: none"> When creating an instance of a standard NetSuite record type, set this value by using the record.Type enum. When creating an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
<code>options.isDynamic</code>	boolean true false	optional	<p>Determines whether the new record is created in dynamic mode.</p> <ul style="list-style-type: none"> If set to <code>true</code>, the new record is created in dynamic mode. If set to <code>false</code>, the new record is created in standard mode. <p>By default, this value is <code>false</code>.</p> <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	<p>Name-value pairs containing default values of fields in the new record. By default, this value is null. For a list of available record default values, see N/record Default Values in the NetSuite Help Center.</p>	2015.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...

// Start the process of creating a sales order record.
```

```

var objRecord = record.create({
    type: record.Type.SALES_ORDER,
    isDynamic: true,
    defaultValues: {
        entity: 87
    }
});

// Start the process of creating an instance of a custom record type.

var customRecord = record.create({
    type: 'customrecord_feature',
    isDynamic: true
});

...
// Add additional code.

```

record.create.promise(options)

Method Description	Creates a new record asynchronously. Note: For information about the parameters and errors thrown for this method, see record.create(options) . For more information on promises, see Promise Object .
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript Object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	<p>The record type. This value determines the Record.type property of the record that is created. This property is read-only on an existing record. Use the following guidelines:</p> <ul style="list-style-type: none"> When creating an instance of a standard NetSuite record type, set this value by using the record.Type enum. 	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> When creating an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	
<code>options.isDynamic</code>	boolean <code>true</code> <code>false</code>	optional	<p>Determines whether the new record is created in dynamic mode.</p> <ul style="list-style-type: none"> If set to <code>true</code>, the new record is created in dynamic mode. If set to <code>false</code>, the new record is created in standard mode. <p>By default, this value is <code>false</code>.</p> <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	<p>Name-value pairs containing default values of fields in the new record.</p> <p>By default, this value is null.</p>	2015.2

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.

...
function createRecord() {

    // Create an instance of a standard record record
    // type.

    var createRecordPromise = record.create.promise({
        type: record.Type.PHONE_CALL,
        isDynamic: true
    });

    // Note: To create an instance of a custom record type,
    // use the record type's string ID instead of the record
    // module's Type enum. For example:
    // type: 'customrecord_feature'

    createRecordPromise.then(function(objRecord) {
        objRecord.setValue({
```

```

        fieldId: 'title',
        value: 'sprint planning'
    });

    var recordId = objRecord.save();

    log.debug({
        title: 'New record saved',
        details: 'New record ID: ' + recordId
    });

    }, function(e) {
        log.error({
            title: 'Unable to create record',
            details: e.name
        });
    });
}
...
// Add additional code.

```

record.delete(options)

Method Description	Deletes a record. <div>  Note: For the promise version of this method, see record.delete.promise(options). Note that promises are only supported in client scripts. </div>
Returns	The internal ID of the deleted record.Record .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 20 usage units Custom records: 4 usage units All other records: 10 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.				
Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The record type. Use the following guidelines: <ul style="list-style-type: none"> When deleting an instance of a standard NetSuite record type, set this value by using the record.Type enum. 	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> When deleting an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	
options.id	number string	required	The internal ID of the record instance to be deleted.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...

// Delete a sales order.

var salesOrderRecord = record.delete({
    type: record.Type.SALES_ORDER,
    id: 88,
});

// Delete an instance of a custom record type with the ID customrecord_feature.

var featureRecord = record.delete({
    type: 'customrecord_feature',
    id: 3,
});
...
// Add additional code.
```

record.delete.promise(options)

Method Description	Deletes a record asynchronously. <div>  Note: For information about the parameters and errors thrown for this method, see record.delete(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 20 usage units

	Custom records: 4 usage units All other records: 10 usage units
Module	N/record Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The record type. Use the following guidelines: <ul style="list-style-type: none"> When deleting an instance of a standard NetSuite record type, set this value by using the record.Type enum. When deleting an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
options.id	number string	required	The internal ID of the record instance to be deleted.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.

// Delete an instance of a standard NetSuite record
...

function deleteRecord() {

    var deleteRecordPromise = record.delete.promise({
        type: record.Type.PHONE_CALL,
        id: 109
    });

    // To delete an instance of a custom record type, use
    // the string ID in the type field. For example:
    // type: 'customrecord_feature'

    deleteRecordPromise.then(function() {

        log.debug({
            title: 'Success',
```

```

        details: 'Record successfully deleted'
    });

    // Add any other needed code that should execute
    // after the record is deleted.

    }, function(e) {
        log.error({
            title: 'Unable to delete record',
            details: e.name
        });
    });
}

...
// Add additional code

```

record.detach(options)

Method Description	Detaches a record from another record.
	 Note: For the promise version of this method, see record.detach.promise(options) . Note that promises are only supported in client scripts.
Returns	void
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.record</code>	record.Record	required	The record to be detached.	2015.2
<code>options.record.type</code>	string	required	The type of record to be detached. Set this value using the record.Type enum.	2015.2
<code>options.record.id</code>	number string	required	The ID of the record to be detached.	2015.2
<code>options.from</code>	record.Record	required	The destination record that <code>options.record</code> should be detached from.	2015.2
<code>options.from.type</code>	string	required	The type of the destination.	2015.2

Parameter	Type	Required / Optional	Description	Since
			Set this value using the record.Type enum.	
<code>options.from.id</code>	number string	required	The ID of the destination.	2015.2
<code>options.attributes</code>	Object	optional	Name-value pairs containing default values of fields in the new record. By default, this value is null.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
record.detach({
  record: {
    type: 'file',
    id: '200'
  },
  from: {
    type: 'customer',
    id: '90'
  }
})
...
// Add additional code.
```

record.detach.promise(options)

Method Description	Detaches a record from another record asynchronously. <div>  Note: For information about the parameters and errors thrown for this method, see record.detach(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/record Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.record</code>	record.Record	required	The record to be detached.	2015.2
<code>options.record.type</code>	string	required	The type of record to be detached. Set this value using the record.Type enum.	2015.2
<code>options.record.id</code>	number string	required	The ID of the record to be detached.	2015.2
<code>options.from</code>	record.Record	required	The destination record that <code>options.record</code> should be detached from.	2015.2
<code>options.from.type</code>	string	required	The type of the destination. Set this value using the record.Type enum.	2015.2
<code>options.from.id</code>	number string	required	The ID of the destination.	2015.2
<code>options.attributes</code>	Object	optional	Name-value pairs containing default values of fields in the new record. By default, this value is null.	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.
...
function detachRecord() {

    var detachRecordPromise = record.detach.promise({
        record: {
            type: record.Type.CONTACT,
            id: '98'
        },
        from: {
            type: record.Type.OPPORTUNITY,
            id: '16'
        }
    });

    detachRecordPromise.then(function() {

        // Add any other needed logic that shouldn't execute until
```

```

    // after the contact record is detached from the opportunity.

    log.debug({
      title: 'Record updated',
      details: 'Contact record detached'
    });

    }, function(e) {
      log.error({
        title: e.name,
        details: e.message
      });
    });
  });
}
...
// Add additional code.

```

record.load(options)

Method Description	Loads an existing record. <div>  Note: For the promise version of this method, see record.load.promise(options). Note that promises are only supported in client scripts. </div>
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	required	<p>The record type. Use the following guidelines:</p> <ul style="list-style-type: none"> When loading an instance of a standard NetSuite record type, set this value by using the record.Type enum. When loading an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	number	required	The internal ID of the existing record instance in NetSuite. The internal ID of the record is displayed on the list page for the record type.	
<code>options.isDynamic</code>	boolean true false	optional	<p>Determines whether the record is loaded in dynamic mode.</p> <ul style="list-style-type: none"> ■ If set to <code>true</code>, the record is loaded in dynamic mode. ■ If set to <code>false</code>, the record is loaded in standard mode. <p>By default, this value is <code>false</code>.</p> <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	<p>Name-value pairs containing default values of fields in the new record.</p> <p>By default, this value is null.</p> <p>For a list of available record default values, see N/record Default Values in the NetSuite Help Center.</p>	

Errors

Error Code	Thrown If
<code>SSS_MISSING_REQD_ARGUMENT</code>	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...

// Load a sales order.

var objRecord = record.load({
  type: record.Type.SALES_ORDER,
  id: 157,
  isDynamic: true,
});

// Load an instance of a custom record type with the ID customrecord_feature.

var newFeatureRecord = record.load({
  type: 'customrecord_feature',
  id: 1,
  isDynamic: true
});
...
// Add additional code.
```

record.load.promise(options)

Method Description	Loads an existing record asynchronously. <div>  Note: For information about the parameters and errors thrown for this method, see record.load(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	required	The record type. Use the following guidelines: <ul style="list-style-type: none"> When loading an instance of a standard NetSuite record type, set this value by using the record.Type enum. When loading an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
<code>options.id</code>	number	required	The internal ID of the existing record instance in NetSuite. The internal ID of the record is displayed on the list page for the record type.	
<code>options.isDynamic</code>	boolean true false	optional	Determines whether the record is loaded in dynamic mode. <ul style="list-style-type: none"> If set to <code>true</code>, the record is loaded in dynamic mode. If set to <code>false</code>, the record is loaded in standard mode. By default, this value is <code>false</code> . <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	Name-value pairs containing default values of fields in the new record.	

Parameter	Type	Required / Optional	Description	Since
			By default, this value is null.	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
function loadRecord() {

    // Load an instance of a standard NetSuite record
    // type.

    var loadRecordPromise = record.load.promise({
        type: record.Type.PHONE_CALL,
        id: 712
    });

    // Note: To load an instance of a custom record type,
    // use the record type's string ID. For example:
    // type: 'customrecord_feature'

    loadRecordPromise.then(function(objRecord) {
        objRecord.setValue({
            fieldId: 'message',
            value: 'We will start the call with a restrospective.'
        });

        var recordId = objRecord.save();

        // Add any other needed logic that shouldn't execute
        // until after the record is instantiated.

        log.debug({
            title: 'Record updated',
            details: 'Updated record ID: ' + recordId
        });

    }, function(e) {
        log.error({
            title: 'Unable to load record',
            details: e.name
        });
    });

    ...
    // Add additional code.
}
```

record.submitFields(options)

Method Description	<p>Updates and submits one or more body fields on an existing record in NetSuite, and returns the internal ID of the parent record.</p> <p>When you use this method, you do not need to load or submit the parent record. You can use this method to edit and submit the following:</p> <ul style="list-style-type: none"> Standard body fields that support inline editing (direct list editing). For more information, see the help topic Using Inline Editing. Custom body fields that support inline editing. <p>You cannot use this method to edit and submit the following:</p> <ul style="list-style-type: none"> Select fields Sublist line item fields Subrecord fields (for example, address fields) <div>  Note: For the promise version of this method, see record.submitFields.promise(options). Note that promises are only supported in client scripts. </div>
Returns	The internal ID of the parent record.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	required	<p>The record type. Use the following guidelines:</p> <ul style="list-style-type: none"> When working with an instance of a standard NetSuite record type, set this value by using the record.Type enum. When working with an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
<code>options.id</code>	number string	required	The internal ID of the existing record instance in NetSuite.	2015.2
<code>options.values</code>	Object	required	<p>The ID-value pairs for each field you want to edit and submit. The value type must correspond to the field type being set. For example:</p>	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> Text, Radio, Select and Multi-Select fields accept string values. Checkbox fields accept Boolean values. Date and DateTime fields accept Date values. Integer, Float, Currency and Percent fields accept number values. 	
options.options	Object	optional	Additional options to set for the record.	2015.2
options.options.enableSourcing	boolean true false	optional	Indicates whether to enable sourcing during the record update. By default, this value is <code>true</code> .	2015.2
options.options.ignoreMandatoryFields	boolean true false	optional	Indicates whether to ignore mandatory fields during record submission. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...

// Submit a new value for a sales order's memo field.

var id = record.submitFields({
  type: record.Type.SALES_ORDER,
  id: 1,
  values: {
    memo: 'ABC'
  },
  options: {
    enableSourcing: false,
    ignoreMandatoryFields : true
  }
});

// Submit a new value for a field on an instance of the 'customrecord_book' custom record type.

var otherId = record.submitFields({
  type: 'customrecord_book',
  id: '4',
  values: {
    'custrecord_rating': '2'
  }
})
```

```
});

...
// Add additional code.
```

record.submitFields.promise(options)

Method Description	<p>Updates and submits one or more body fields asynchronously on an existing record in NetSuite, and returns the internal ID of the parent record.</p> <p>When you use this method, you do not need to load or submit the parent record. You can use this method to edit and submit the following:</p> <ul style="list-style-type: none"> Standard body fields that support inline editing (direct list editing). For more information, see the help topic Using Inline Editing. Custom body fields that support inline editing. <p>You cannot use this method to edit and submit the following:</p> <ul style="list-style-type: none"> Select fields Sublist line item fields Subrecord fields (for example, address fields) <div>  Note: For information about the parameters and errors thrown for this method, see record.submitFields(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	Client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other records: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	<p>The record type. Use the following guidelines:</p> <ul style="list-style-type: none"> When working with an instance of a standard NetSuite record type, set this value by using the record.Type enum. When working with an instance of a custom record type, set this value by using the custom record type's string ID. For help finding this ID, see the help topic Custom Record. 	2015.2
options.id	number string	required	The internal ID of the existing record instance in NetSuite.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.values	Object	required	The ID-value pairs for each field you want to edit and submit.	2015.2
options.options	Object	optional	Additional options to set for the record.	2015.2
options.options.enableSourcing	boolean true false	optional	Indicates whether to enable sourcing during the record update. By default, this value is <code>true</code> .	2015.2
options.options.ignoreMandatoryFields	boolean true false	optional	Indicates whether to ignore mandatory fields during record submission. By default, this value is <code>false</code> .	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code
...
function submitFields() {

    var submitFieldsPromise = record.submitFields.promise({
        type: record.Type.PHONE_CALL,
        id: 171,
        values: {
            title: 'Sprint 3 planning'
        },
    });

    submitFieldsPromise.then(function(recordId) {

        // Add any needed logic that shouldn't execute until
        // after the new value is submitted.

        log.debug({
            title: 'Record updated',
            details: 'Id of updated record: ' + recordId
        });

    }, function(e) {
        log.error({
            title: e.name,
            details: e.message
        });
    });

}
...
// Add additional code
```

record.transform(options)

Method Description	<p>Transforms a record from one type into another, using data from an existing record. You can use this method to automate order processing, creating item fulfillment transactions and invoices off of orders. For a list of supported transformations, see Supported Transformation Types.</p> <div>  Note: For the promise version of this method, see record.transform.promise(options). Note that promises are only supported in client scripts. </div>
Returns	record.Record
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	Transaction records: 10 usage units Custom records: 2 usage units All other record types: 5 usage units
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.fromType</code>	string	required	The record type of the existing record instance being transformed. This value sets the Record.type property for the record. This property is read-only and cannot be changed after the record is loaded. Set this value using the record.Type .	2015.2
<code>options.fromId</code>	number	required	The internal ID of the existing record instance being transformed.	2015.2
<code>options.toType</code>	string	required	The record type of the record returned when the transformation is complete.	2015.2
<code>options.isDynamic</code>	boolean true false	optional	Determines whether the new record is created in dynamic mode. <ul style="list-style-type: none"> ■ If set to <code>true</code>, the new record is created in dynamic mode. ■ If set to <code>false</code>, the new record is created in standard mode. By default, this value is <code>false</code> . <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	Name-value pairs containing default values of fields in the new record. By default, this value is null.	2015.2

Parameter	Type	Required / Optional	Description	Since
			For a list of available record default values, see N/record Default Values in the NetSuite Help Center.	

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.transform({
    fromType: record.Type.CUSTOMER,
    fromId: 107,
    toType: record.Type.SALES_ORDER,
    isDynamic: true,
});
...
// Add additional code.
```

Supported Transformation Types

Original Record Type	Transformed Record Type
Build/Assembly	Assembly Build
Assembly Build	Assembly Unbuild
Cash Sale	Cash Sale
Customer	Cash Sale
Customer	Customer Payment
Customer	Quote
Customer	Invoice
Customer	Opportunity
Customer	Sales Order
Employee	Expense Report
Employee	Time
Quote	Cash Sale
Quote	Invoice
Quote	Sales Order
Invoice	Credit Memo
Invoice	Customer Payment

Original Record Type	Transformed Record Type
Invoice	Return Authorization
Lead	Opportunity
Opportunity	Cash Sale
Opportunity	Quote
Opportunity	Invoice
Opportunity	Sales Order
Prospect	Quote
Prospect	Opportunity
Prospect	Sales Order
Purchase Order	Item Receipt
Purchase Order	Vendor Bill
Purchase Order	Vendor Return Authorization
Return Authorization	Cash Refund
Return Authorization	Credit Memo
Return Authorization	Item Receipt
Return Authorization	Revenue Commitment Reversal
	<div>  Note: The return authorization must be approved and received for this transform to work. </div>
Sales Order	Cash Sale
Sales Order	Invoice
Sales Order	Item Fulfillment
Sales Order	Return Authorization
Sales Order	Revenue Commitment
Transfer Order	Item Fulfillment
Transfer Order	Item Receipt
Vendor	Purchase Order
Vendor	Vendor Bill
Vendor Bill	Vendor Credit
Vendor Bill	Vendor Payment
Vendor Bill	Vendor Return Authorization
Vendor Return Authorization	Item Fulfillment
Vendor Return Authorization	Vendor Credit
Work Order	Assembly Build

record.transform.promise(options)

Method Description	<p>Transforms a record from one type into another asynchronously, using data from an existing record.</p> <p>You can use this method to automate order processing, creating item fulfillment transactions and invoices off of orders.</p> <p>For a list of supported transformations, see Supported Transformation Types.</p> <div>  Note: For information about the parameters and errors thrown for this method, see record.transform(options). For more information on promises, see Promise Object. </div>
Returns	Promise
Supported Script Types	<p>Client-side scripts</p> <p>For more information, see SuiteScript 2.0 Client Script Type.</p>
Governance	<p>Transaction records: 10 usage units</p> <p>Custom records: 2 usage units</p> <p>All other record types: 5 usage units</p>
Module	N/record Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.fromType</code>	string	required	<p>The record type of the existing record instance being transformed.</p> <p>This value sets the Record.type property for the record. This property is read-only and cannot be changed after the record is loaded.</p> <p>Set this value using the record.Type.</p>	2015.2
<code>options.fromId</code>	number	required	The internal ID of the existing record instance being transformed.	2015.2
<code>options.toType</code>	string	required	The record type of the record returned when the transformation is complete.	2015.2
<code>options.isDynamic</code>	boolean true false	optional	<p>Determines whether the new record is created in dynamic mode.</p> <ul style="list-style-type: none"> ■ If set to <code>true</code>, the new record is created in dynamic mode. ■ If set to <code>false</code>, the new record is created in standard mode. <p>By default, this value is <code>false</code>.</p> <div>  Note: For additional information on standard and dynamic mode, see record.Record and SuiteScript 2.0 – Standard and Dynamic Modes. </div>	2015.2
<code>options.defaultValues</code>	Object	optional	<p>Name-value pairs containing default values of fields in the new record.</p> <p>By default, this value is null.</p>	2015.2

Errors

Error Code	Thrown If
SSS_MISSING_REQD_ARGUMENT	A required argument is missing or undefined.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
// Add additional code.
...
function transformRecord() {

    var transformRecordPromise = record.transform.promise({
        fromType: record.Type.ESTIMATE,
        fromId: 25,
        toType: record.Type.SALES_ORDER,
        isDynamic: true,
    });

    transformRecordPromise.then(function(recordObject) {

        var recordId = recordObject.save();

        // Add any other needed logic that shouldn't execute until
        // after the record is transformed.

        log.debug({
            title: 'Record saved',
            details: 'Id of new record: ' + recordId
        });

    }, function(e) {

        log.error({
            title: e.name,
            details: e.message
        });

    });

}
...
// Add additional code.
```

record.Type

Enum Description	Enumeration that holds the string values for supported record types. This enum is used to set the value of the Record.type property in cases where you are working with an instance of a standard NetSuite record type. (If you are working with an instance of a custom record type, you set the Record.type property by using the custom record type's string ID. For more help finding this ID, see the help topic Custom Record .)
-------------------------	--

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/record Module
Since	2015.2

Values

<ul style="list-style-type: none"> ■ ACCOUNT ■ ACCOUNTING_BOOK ■ ACCOUNTING_CONTEXT ■ ACCOUNTING_PERIOD ■ ADV_INTER_COMPANY_JOURNAL_ENTRY ■ ALLOCATION_SCHEDULE ■ AMORTIZATION_SCHEDULE ■ AMORTIZATION_TEMPLATE ■ ASSEMBLY_BUILD ■ ASSEMBLY_ITEM ■ ASSEMBLY_UNBUILD ■ BILLING_ACCOUNT ■ BILLING_CLASS ■ BILLING_RATE_CARD ■ BILLING_REVENUE_EVENT ■ BILLING_SCHEDULE ■ BIN ■ BIN_TRANSFER ■ BIN_WORKSHEET ■ BLANKET_PURCHASE_ORDER ■ BOM ■ BOM_REVISION ■ BULK_OWNERSHIP_TRANSFER ■ BUNDLE_INSTALLATION_SCRIPT ■ CALENDAR_EVENT ■ CAMPAIGN ■ CAMPAIGN_RESPONSE ■ CAMPAIGN_TEMPLATE ■ CASH_REFUND ■ CASH_SALE ■ CHARGE ■ CHARGE_RULE ■ CHECK 	<ul style="list-style-type: none"> ■ EMAIL_TEMPLATE ■ EMPLOYEE ■ ENTITY_ACCOUNT_MAPPING ■ ESTIMATE ■ EXPENSE_CATEGORY ■ EXPENSE_REPORT ■ FAIR_VALUE_PRICE ■ FIXED_AMOUNT_PROJECT_REVENUE_RULE ■ FOLDER ■ FULFILLMENT_REQUEST ■ GENERIC_RESOURCE ■ GIFT_CERTIFICATE ■ GIFT_CERTIFICATE_ITEM ■ GLOBAL_ACCOUNT_MAPPING ■ GLOBAL_INVENTORY_RELATIONSHIP ■ GOVERNMENT_ISSUED_ID_TYPE ■ HCM_JOB ■ INBOUND_SHIPMENT ■ INTERCOMP_ALLOCATION_SCHEDULE ■ INTER_COMPANY_JOURNAL_ENTRY ■ INTER_COMPANY_TRANSFER_ORDER ■ INVENTORY_ADJUSTMENT ■ INVENTORY_COST_REVALUATION ■ INVENTORY_COUNT ■ INVENTORY_DETAIL ■ INVENTORY_ITEM ■ INVENTORY_NUMBER ■ INVENTORY_TRANSFER ■ INVOICE ■ ISSUE 	<ul style="list-style-type: none"> ■ POSITION ■ PRICE_BOOK ■ PRICE_LEVEL ■ PRICE_PLAN ■ PRICING_GROUP ■ PROJECT_EXPENSE_TYPE ■ PROJECT_TASK ■ PROJECT_TEMPLATE ■ PROMOTION_CODE ■ PROSPECT ■ PURCHASE_CONTRACT ■ PURCHASE_ORDER ■ PURCHASE_REQUISITION ■ RATE_PLAN ■ REALLOCATE_ITEM ■ RECEIVE_INBOUND_SHIPMENT ■ RESOURCE_ALLOCATION ■ RESTLET ■ RETURN_AUTHORIZATION ■ REVENUE_ARRANGEMENT ■ REVENUE_COMMITMENT ■ REVENUE_COMMITMENT_REVERSAL ■ REVENUE_PLAN ■ REV_REC_SCHEDULE ■ REV_REC_TEMPLATE ■ SALES_ORDER ■ SALES_TAX_ITEM ■ SCHEDULED_SCRIPT ■ SCHEDULED_SCRIPT_INSTANCE ■ SCRIPT_DEPLOYMENT ■ SERIALIZED_ASSEMBLY_ITEM ■ SERIALIZED_INVENTORY_ITEM ■ SERVICE_ITEM
--	---	---

<ul style="list-style-type: none"> ■ CLASSIFICATION ■ CLIENT_SCRIPT ■ CMS_CONTENT ■ CMS_CONTENT_TYPE ■ COMMERCE_CATEGORY ■ COMPETITOR ■ CONSOLIDATED_EXCHANGE_RATE ■ CONTACT ■ CONTACT_CATEGORY ■ CONTACT_ROLE ■ COST_CATEGORY ■ COUPON_CODE ■ CREDIT_CARD_CHARGE ■ CREDIT_CARD_REFUND ■ CREDIT_MEMO ■ CURRENCY ■ CUSTOMER ■ CUSTOMER_CATEGORY ■ CUSTOMER_DEPOSIT ■ CUSTOMER_MESSAGE ■ CUSTOMER_PAYMENT ■ CUSTOMER_PAYMENT_AUTHORIZATION ■ CUSTOMER_REFUND ■ CUSTOMER_STATUS ■ CUSTOMER_SUBSIDIARY_RELATIONSHIP ■ CUSTOM_RECORD ■ CUSTOM_TRANSACTION ■ DEPARTMENT ■ DEPOSIT ■ DEPOSIT_APPLICATION ■ DESCRIPTION_ITEM ■ DISCOUNT_ITEM ■ DOWNLOAD_ITEM ■ DRIVERS_LICENSE 	<ul style="list-style-type: none"> ■ ISSUE_PRODUCT ■ ISSUE_PRODUCT_VERSION ■ ITEM_ACCOUNT_MAPPING ■ ITEM_DEMAND_PLAN ■ ITEM_FULFILLMENT ■ ITEM_GROUP ■ ITEM_LOCATION_CONFIGURATION ■ ITEM_RECEIPT ■ ITEM_REVISION ■ ITEM_SUPPLY_PLAN ■ JOB ■ JOB_REQUISITION ■ JOB_STATUS ■ JOB_TYPE ■ JOURNAL_ENTRY ■ KIT_ITEM ■ KUDOS ■ LABOR_BASED_PROJECT_REVENUE_RULE ■ LEAD ■ LOCATION ■ LOT_NUMBERED_ASSEMBLY_ITEM ■ LOT_NUMBERED_INVENTORY_ITEM ■ MANUFACTURING_COST_TEMPLATE ■ MANUFACTURING_OPERATION_TASK ■ MANUFACTURING_ROUTING ■ MAP_REDUCE_SCRIPT ■ MARKUP_ITEM ■ MASSUPDATE_SCRIPT ■ MERCHANDISE_HIERARCHY_LEVEL ■ MERCHANDISE_HIERARCHY_NODE ■ MERCHANDISE_HIERARCHY_VERSION ■ MESSAGE ■ MFG_PLANNED_TIME ■ NEXUS ■ NON_INVENTORY_ITEM ■ NOTE ■ NOTE_TYPE ■ OPPORTUNITY ■ ORDER_SCHEDULE 	<ul style="list-style-type: none"> ■ SHIP_ITEM ■ SOLUTION ■ STATISTICAL_JOURNAL_ENTRY ■ STORE_PICKUP_FULFILLMENT ■ SUBSCRIPTION ■ SUBSCRIPTION_CHANGE_ORDER ■ SUBSCRIPTION_LINE ■ SUBSCRIPTION_PLAN ■ SUBSIDIARY ■ SUBTOTAL_ITEM ■ SUITELET ■ SUPPORT_CASE ■ TASK ■ TAX_ACCT ■ TAX_GROUP ■ TAX_PERIOD ■ TAX_TYPE ■ TERM ■ TERMINATION_REASON ■ TIME_BILL ■ TIME_OFF_CHANGE ■ TIME_OFF_PLAN ■ TIME_OFF_REQUEST ■ TIME_OFF_RULE ■ TIME_OFF_TYPE ■ TOPIC ■ TRANSFER_ORDER ■ UNITS_TYPE ■ USAGE ■ USEREVENT_SCRIPT ■ VENDOR ■ VENDOR_BILL ■ VENDOR_CATEGORY ■ VENDOR_CREDIT ■ VENDOR_PAYMENT ■ VENDOR_RETURN_AUTHORIZATION ■ WEBSITE ■ WORKFLOW_ACTION_SCRIPT ■ WORK_ORDER ■ WORK_ORDER_CLOSE ■ WORK_ORDER_COMPLETION ■ WORK_ORDER_ISSUE ■ WORKPLACE
--	--	---

	<ul style="list-style-type: none"> ■ ORGANIZATION_VALUE ■ OTHER_CHARGE_ITEM ■ OTHER_GOVERNMENT_ISSUED_ID ■ OTHER_NAME ■ PARTNER ■ PARTNER_CATEGORY ■ PASSPORT ■ PAYCHECK_JOURNAL ■ PAYMENT_ITEM ■ PAYMENT_METHOD ■ PAYROLL_ITEM ■ PERIOD_END_JOURNAL ■ PHONE_CALL ■ PORTLET 	
--	---	--

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
// Add additional code.
...
var objRecord = record.delete({
    type: record.Type.SALES_ORDER,
    id: 128
});
...
// Add additional code.
```

N/redirect Module

Use the redirect module to customize navigation within NetSuite by setting up a redirect URL that resolves to a NetSuite resource or external URL. You can redirect users to one of the following:

- URL
- Suitelet
- Record
- Task link
- Saved search
- Unsaved search



Note: Suitelets, beforeLoad user events, and synchronous afterSubmit user events are supported. This module does not support beforeSubmit and asynchronous afterSubmit user events.

- [N/redirect Module Members](#)

N/redirect Module Script Sample

N/redirect Module Members

Member Type	Name	Return Type	Supported Script Types	Description
Method	<code>redirect.redirect(options)</code>	void	Suitelets, beforeLoad user events, and synchronous afterSubmit user events	Redirects to the URL of a Suitelet that is available externally (available without login).
	<code>redirect.toRecord(options)</code>	void	Suitelets, beforeLoad user events, and synchronous afterSubmit user events	Redirects to a NetSuite record.
	<code>redirect.toSavedSearch(options)</code>	void	afterSubmit user events	Redirects to a saved search.
	<code>redirect.toSavedSearchResult(options)</code>	void	afterSubmit user events	Redirects to a saved search result.
	<code>redirect.toSearch(options)</code>	void	afterSubmit user events	Redirects to search.
	<code>redirect.toSearchResult(options)</code>	void	afterSubmit user events	Redirects to search results.
	<code>redirect.toSuitelet(options)</code>	void	Suitelets, beforeLoad user events, and synchronous afterSubmit user events	Redirects to a Suitelet.
	<code>redirect.toTaskLink(options)</code>	void	Suitelets, beforeLoad user events, and synchronous afterSubmit user events	Redirects to a tasklink.

N/redirect Module Script Sample

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

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
/**
 * @ApiVersion 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();
});

```

redirect.redirect(options)

Method Description	Method used to set the redirect to the URL of a Suitelet that is available externally (Suitelets set to Available Without Login on the Script Deployment page).
Returns	Void
Supported Script Types	Suitelets, beforeLoad user events, and synchronous afterSubmit user events For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/redirect Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.url	string	required	The URL of a Suitelet that is available externally  Note: For an external URL, Available without Login must be enabled on the Script Deployment page for the Suitelet.
options.parameters	Object	optional	Contains additional URL parameters as key/value pairs.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```

//Add additional code
...
redirect.redirect({
    url: /app/site/hosting/scriptlet.nl?script=130&deploy=1,

```

```

    parameters: {'custparam_test': 'helloWorld'}
  });
  ...
  //Add additional code

```

redirect.toRecord(options)

Method Description	Method used to set the redirect URL to a specific NetSuite record. Note: If you redirect a user to a record, the record must first exist in NetSuite. If you want to redirect a user to a new record, you must first create and submit the record before redirecting them. You must also ensure that any required fields for the new record are populated before submitting the record.
Returns	Void
Supported Script Types	Suitelets, beforeLoad user events, and synchronous afterSubmit user events For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/redirect Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The internal id of the target record.
options.type	string	required	The type of record.
options.isEditMode	boolean true false	optional	Determines whether to return a URL for the record in edit mode or view mode. If set to true, returns the URL to an existing record in edit mode. The default value is false - returns the URL to a record in view mode.
options.parameters	Object	optional	Contains additional URL parameters as key/value pairs.

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```

//Add additional code
...
redirect.toRecord({
  type : record.Type.TASK,
  id : taskRecordId,
  parameters: {'custparam_test': 'helloWorld'}
});

```

```
...
//Add additional code
```

redirect.toSavedSearch(options)

Method Description	Method used to load an existing saved search and redirect to the populated search definition page.
Returns	Void
Supported Script Types	afterSubmit user event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Governance	5 units
Module	N/redirect Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	number	required	Internal ID of the search. The internal ID is available only when the search is either loaded with search.load(options) or after is has been saved with Search.save() . Typical values are 55 or 234 or 87, not a value like <code>customsearch_mysearch</code> . Any ID prefixed with <code>customsearch</code> is a script ID, not the internal system ID for a search.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```
//Add additional code
...
redirect.toSavedSearch({id: 234});
...
//Add additional code
```

redirect.toSavedSearchResult(options)

Method Description	Method used to redirect a user to a search results page for an existing saved search.
Returns	Void
Supported Script Types	afterSubmit user event scripts For more information, see SuiteScript 2.0 User Event Script Type .

Governance	5 units
Module	N/redirect Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	number	required	Internal ID of the search. The internal ID is available only when the search is either loaded with search.load(options) or after it has been saved with Search.save() . Typical values are 55 or 234 or 87, not a value like <code>customsearch_mysearch</code> . Any ID prefixed with <code>customsearch</code> is a script ID, not the internal system ID for a search.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```
//Add additional code
...
redirect.toSavedSearchResult({id: 234});
...
//Add additional code
```

redirect.toSearch(options)

Method Description	Method used to redirect a user to an on demand search built in SuiteScript. This method loads a search into the session, and then redirects to a URL that loads the search definition page.
Returns	Void
Supported Script Types	afterSubmit user event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Governance	None
Module	N/redirect Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.search	search.Search	required	

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```
var column = ['internalid'];
var filter = [{"mainline", "is", "T"}];
var ourNewSearch = search.create({
    id: 'customsearch_test',
    type: search.Type.SALES_ORDER,
    title: 'My Generated Search',
    columns: column,
    filters: filter
});
redirect.toSearch({
    search: ourNewSearch
});
```

redirect.toSearchResult(options)

Method Description	Method used to redirect a user to a search results page. For example, the results from an on demand search created with the N/search Module , or a loaded search that you modified but did not save.
Returns	Void
Supported Script Types	afterSubmit user event scripts For more information, see SuiteScript 2.0 User Event Script Type .
Governance	None
Module	N/redirect Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.Search	search.Search	required	

redirect.toSuitelet(options)

Method Description	Method used to redirect the user to a Suitelet. For more information about Suitelets, see SuiteScript 2.0 Suitelet Script Type .
Returns	Void
Supported Script Types	Suitelets, beforeLoad user events, and synchronous afterSubmit user events For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/redirect Module

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.scriptId	string	required	The script ID for the Suitelet.
options.deploymentId	string	required	The deployment ID for the Suitelet.
options.isExternal	boolean true false	optional	The default value is <code>false</code> – indicates an external Suitelet URL.
options.parameters	Object	optional	Contains additional URL parameters as key/value pairs.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```
//Add additional code
...
redirect.toSuitelet({
    scriptId: 31 ,
    deploymentId: 1,
    parameters: {'custparam_test':'helloWorld'}
});
...
//Add additional code
```

redirect.toTaskLink(options)

Method Description	Method used to redirect a user to a tasklink.
Returns	Void
Supported Script Types	Suitelets, beforeLoad user events, and synchronous afterSubmit user events For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/redirect Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The taskId for a tasklink

Parameter	Type	Required / Optional	Description
			For a list of supported task IDs, see the help topic Task IDs .
options.parameters	Object	optional	Contains additional URL parameters as key/value pairs.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/redirect Module Script Sample](#).

```
//Add additional code
...
redirect.toTaskLink({
  id: ADMI_SHIPPING ,
  parameters: {'custparam_test': 'helloWorld'}
});
...
//Add additional code
```

N/render Module

The render module encapsulates functionality for printing, PDF creation, form creation from templates, and email creation from templates.



Note: Direct manipulation of the print URL is **not** supported.

- [N/render Module Members](#)
- [EmailMergeResult Object Members](#)
- [TemplateRenderer Object Members](#)
- [N/render Module Script Sample](#)

N/render Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	render.EmailMergeResult	Object	Server-side scripts	Encapsulates an email merge result
	render.TemplateRenderer	Object	Server-side scripts	Encapsulates a template engine object that produces HTML and PDF printed forms utilizing advanced PDF/HTML template capabilities
Method	render.bom(options)	file.File	Server-side scripts	Creates a PDF or HTML file object containing a bill of materials

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>render.create()</code>	<code>render.TemplateRenderer</code>	Server-side scripts	Creates a <code>render.TemplateRenderer</code> object
	<code>render.mergeEmail(options)</code>	<code>render.EmailMergeResult</code>	Server-side scripts	Creates a <code>render.EmailMergeResult</code> object
	<code>render.packingSlip(options)</code>	<code>file.File</code>	Server-side scripts	Creates a PDF or HTML file object containing a packing slip
	<code>render.pickingTicket(options)</code>	<code>file.File</code>	Server-side scripts	Creates a PDF or HTML file object containing a picking ticket
	<code>render.statement(options)</code>	<code>file.File</code>	Server-side scripts	Creates a PDF or HTML file object containing a statement
	<code>render.transaction(options)</code>	<code>file.File</code>	Server-side scripts	Creates a PDF or HTML file object containing a transaction
	<code>render.xmlToPdf(options)</code>	<code>file.File</code>	Server-side scripts	Passes XML to the BFO tag library (which is stored by NetSuite), and returns a PDF file
Enum	<code>render.DataSource</code>	enum	Server-side scripts	Holds the string values for supported data source types
	<code>render.PrintMode</code>	enum	Server-side scripts	Holds the string values for supported print output types

EmailMergeResult Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>EmailMergeResult.body</code>	string (read-only)	Server-side scripts	The body of the email distribution in string format
	<code>EmailMergeResult.subject</code>	string (read-only)	Server-side scripts	The subject of the email distribution in string format

TemplateRenderer Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>TemplateRenderer.addCustomDataSource(options)</code>	void	Server-side scripts	Adds an XML file or JSON object to an advanced template as a custom data source
	<code>TemplateRenderer.addRecord(options)</code>	void	Server-side scripts	Binds a record to a template variable

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	TemplateRenderer.addSearchResults(options)	void	Server-side scripts	Binds a search result to a template variable
	TemplateRenderer.renderAsPdf()	Object	Server-side scripts	Uses an advanced template to produce a PDF printed form
	TemplateRenderer.renderPdfToResponse()	void	Server-side scripts	Renders PDF template content as a server response
	TemplateRenderer.renderAsString()	string	Server-side scripts	Returns template content in string form
	TemplateRenderer.setTemplateById(options)	void	Server-side scripts	Sets the template using the internal ID
	TemplateRenderer.setTemplateByScriptId(options)	void	Server-side scripts	Sets the template using the script ID
	TemplateRenderer.renderToResponse(options)	void	Server-side scripts	Renders HTML template content as a server response
Property	TemplateRenderer.templateContent	string	Server-side scripts	Content of template

N/render Module Script Sample

Note: These sample scripts use the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following example generates a PDF file from a raw XML string.

```
/**
 * @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();
});
```

The following example renders a transaction record into an HTML page.

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();
});
```

The following example renders an invoice into a PDF file using an XML template in the file cabinet. This example requires the Advanced PDF/HTML Templates feature.

```
/**
 *@NApiVersion 2.x
 */
// This example 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 example, 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.

The following example renders search results into a PDF file.

Note: This sample script uses the `define` function. You cannot use [On Demand Debugging](#) to step through a `define` function. You must use [Deployed Debugging](#) to step through this script.

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

```
// This example shows how to render search results into a PDF file.
// Note that this sample is a Suitelet, so it cannot be run in the debugger.
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',
            searchResults: results
        });

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

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

render.EmailMergeResult

Object Description	Encapsulates an email merge result. Use render.mergeEmail(options) to create and return this object. For a complete list of this object's properties, see EmailMergeResult Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var mergeResult = render.mergeEmail({
  templateId: 1234,
  entity: {
    type: 'employee',
    id: 62
  },
  recipient: {
    type: 'lead',
    id: 41
  },
  supportCaseId: 2,
  transactionId: 271,
  customRecord: null
});
...
//Add additional code
```

EmailMergeResult.body

Property Description	The body of the email distribution in string format
Type	string (read-only)
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Email Body: ',
  details: mergeResultObj.body
});
...
//Add additional code
```

EmailMergeResult.subject

Property Description	The subject of the email distribution in string format
Type	string (read-only)
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
log.debug({
  title: 'Email Subject: ',
  details: mergeResultObj.subject
});
...

//Add additional code
```

render.TemplateRenderer

Object Description	Encapsulates a template engine object that produces HTML and PDF printed forms utilizing advanced PDF/HTML template capabilities. The template engine object includes methods that pass in a template as string to be interpreted by FreeMarker, and render interpreted content in your choice of two different formats: as HTML output to an nlobjResponse object, or as XML string that can be passed to render.xmlToPdf(options) to produce a PDF. This object is available when the Advanced PDF/HTML Templates feature is enabled. For a complete list of this object's methods and properties, see TemplateRenderer Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
//Advanced PDF/HTML Templates feature must be enabled
...
var xmlTplFile = file.load('Templates/PDF Templates/invoicePDFTemplate.xml');
```

```

var myFile = render.create();
myFile.templateContent = xmlTplFile.getContents();
myFile.addRecord('record', record.load({
    type: record.Type.INVOICE,
    id: 37
}));
var invoicePdf = myFile.renderAsPdf();
...
//Add additional code

```

TemplateRenderer.addCustomDataSource(options)

Method Description	Adds XML or JSON as custom data source to an advanced PDF/HTML template
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.alias	string	required	Data source alias
options.format	render.DataSource	required	Data format
options.data	Object Document string	required	Object, document, or string

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```

//Add additional code
...
var renderer = render.create();

var xmlObj = xml.Parser.fromString(xmlString);
var jsonObj = JSON.parse(jsonString);

renderer.templateContent = "${XML.book.title}<br />${XML.book.chapter[1].title}<br />${JSON.book.title}<br />
${JSON.book.chapter[1].title}<br />${JSON_STR.book.title}<br />${XML_STR.book.title}";

renderer.addCustomDataSource({
    format: render.DataSource.XML_DOC,
    alias: "XML",

```



```

    data: xmlObj
  });
  renderer.addCustomDataSource({
    format: render.DataSource.XML_STRING,
    alias: "XML_STR",
    data: xmlString
  });
  renderer.addCustomDataSource({
    format: render.DataSource.OBJECT,
    alias: "JSON",
    data: jsonObj
  });
  renderer.addCustomDataSource({
    format: render.DataSource.JSON,
    alias: "JSON_STR",
    data: jsonString
  });

  var xml = renderer.renderAsString();
  ...
  //Add additional code

```

TemplateRenderer.addRecord(options)

Method Description	Binds a record to a template variable.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.templateName	string	required	Name of the record object variable referred to in the template
options.record	record.Record object	required	The record to add

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```

//Add additional code
...

```

```

var myContent = renderer.addRecord({
  templateName: 'record',
  record: record.load({
    type: record.Type.CUSTOMER,
    id: 1234
  });
});
...
//Add additional code

```

TemplateRenderer.addSearchResults(options)

Method Description	Binds a search result to a template variable.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.templateName	string	required	Name of the template
options.searchResult	search.Result object	required	The search result to add

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```

//Add additional code
...
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

```

```
);
...
//Add additional code
```

TemplateRenderer.renderAsPdf()

Method Description	Uses the advanced template to produce a PDF printed form
Returns	file.File
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var invoicePdf = renderer.renderAsPdf();
...
//Add additional code
```

TemplateRenderer.renderPdfToResponse()

Method Description	Renders a server response into a PDF file. For example, you can pass in a response to be rendered as a PDF in a browser, or downloaded by a user.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
response	http.ServerResponse	required	Response that will be written to PDF. For example, the response passed from a Suitelet.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var invoicePdf = renderer.renderPdfToResponse({
    response: myServerResponseObj
});
...
//Add additional code
```

TemplateRenderer.renderAsString()

Method Description	Return template content in string form
Returns	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var invoicePdf = renderer.renderAsString();
...
//Add additional code
```

TemplateRenderer.renderToResponse(options)

Method Description	Writes template content to a server response.
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.response	http.ServerResponse	required	Response to write to

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var invoice = render.renderToResponse({
  response: myServerResponseObj
});
...
//Add additional code
```

TemplateRenderer.setTemplateById(options)

Method Description	Sets the template using the internal ID
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	number	required	Internal ID of the template

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var renderer = render.create();
renderer.setTemplateById(3);
```

```
var xml = renderer.renderAsString();
...
//Add additional code
```

TemplateRenderer.setTemplateById(options)

Method Description	Sets the template using the script ID
Returns	Void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.scriptId	string	required	Script ID of the template

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#)

```
//Add additional code
...
var renderer = render.create();
renderer.setTemplateById({
  scriptId: "STDTMPLPRICELIST"
});
var xml = renderer.renderAsString();
...
//Add additional code
```

TemplateRenderer.templateContent

Property Description	Content of template
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
renderer.templateContent = xmlTemplateFile.getContents();
...
//Add additional code
```

render.bom(options)

Method Description	Use this method to create a PDF or HTML object of a bill of material.
Returns	file.File that contains a PDF or HTML document
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.entityId</code>	number	required	The internal ID of the bill of material to print
<code>options.printMode</code>	string	optional	The print output type. Set using the render.PrintMode enum. By default, uses the company/user preference for print output.
<code>options.inCustLocale</code>	boolean true false	optional	Applies when advanced templates are used. Print the document in the customer's locale. If basic printing is used, this parameter is ignored and the transaction form is printed in the customer's locale.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var transactionFile = render.bom({
  entityId: 23,
  printMode: render.PrintMode.HTML,
});
```

```

    inCustLocale: true
  });
  ...
  //Add additional code

```

render.create()

Method Description	<p>Use this method to produce HTML and PDF printed forms with advanced PDF/HTML templates.</p> <p>Creates render.TemplateRenderer.</p> <p>This object includes methods that pass in a template as string to be interpreted by FreeMarker, and render interpreted content in your choice of two different formats: as HTML output to http.ServerResponse, or as XML string that can be passed to render.xmlToPdf(options) to produce a PDF.</p> <div>  Note: To use this method, the Advanced PDF/HTML Templates feature must be enabled. </div>
Returns	render.TemplateRenderer
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```

//Add additional code
...
var renderer = render.create();
...
//Add additional code

```

render.mergeEmail(options)

Method Description	Creates a render.EmailMergeResult object for a mail merge with an existing scriptable email template
Returns	render.EmailMergeResult
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.templateId	number	required	Internal ID of the template
options.entity	RecordRef	required	Entity
options.recipient	RecordRef	required	Recipient
options.customRecord	RecordRef	required	Custom record
options.supportCaseId	number	required	Support case ID
options.transactionId	number	required	Transaction ID

RecordRef

You can use a RecordRef to designate the record to perform the mail merge on.

 **Note:** The RecordRef object encapsulates the type and ID of a particular record instance.

Property	Type	Required / Optional	Description
RecordRef.id	number	required	Internal ID of the record instance
RecordRef.type	string	required	The record type ID

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var myMergeResult = render.mergeEmail({
  templateId: 1234,
  entity: {
    type: 'employee',
    id: 623
  },
  recipient: {
    type: 'lead',
    id: 543
  },
  supportCaseId: 674,
  transactionId: 8987,
  customRecord: {
    type: 'custrecord_rewardpoints',
    id: 5423
  }
});
...
//Add additional code
```

render.packingSlip(options)

Method Description	Use this method to create a PDF or HTML object of a packing slip.
Returns	file.File that contains a PDF or HTML document
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.entityId	number	required	The internal ID of the packing slip to print
options.printMode	string	optional	The print output type. Set using the render.PrintMode enum. By default, uses the company/user preference for print output.
options.formId	number	optional	The packing slip form number
options.fulfillmentId	number	optional	Fulfillment ID number
options.inCustLocale	boolean true false	optional	Applies when advanced templates are used. Print the document in the customer's locale. If basic printing is used, this parameter is ignored and the transaction form is printed in the customer's locale.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var transactionFile = render.packingSlip({
    entityId: 23,
    printMode: render.PrintMode.HTML,
    inCustLocale: true
});
...
//Add additional code
```

render.pickingTicket(options)

Method Description	Use this method to create a PDF or HTML object of a picking ticket.
---------------------------	---

Returns	file.File that contains a PDF or HTML document
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.entityId	number	required	The internal ID of the picking ticket to print
options.printMode	string	optional	The print output type. Set using the render.PrintMode enum. By default, uses the company/user preference for print output.
options.formId	number	optional	The packing slip form number
options.shipgroup	number	optional	Shipping group for the ticket
options.location	number	optional	Location for the ticket
options.inCustLocale	boolean true false	optional	Applies when advanced templates are used. Print the document in the customer's locale. If basic printing is used, this parameter is ignored and the transaction form is printed in the customer's locale.

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var transactionFile = render.pickingTicket({
  entityId: 23,
  printMode: render.PrintMode.HTML,
  inCustLocale: true
});
...
//Add additional code
```

render.statement(options)

Method Description	Use this method to create a PDF or HTML object of a statement.
Returns	file.File that contains a PDF or HTML document
Supported Script Types	Server-side scripts

	For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.entityId	number	required	The internal ID of the statement to print
options.printMode	string	optional	The print output type. Set using the render.PrintMode enum. By default, uses the company/user preference for print output.
options.formId	number	optional	Internal ID of the form to use to print the statement
options.startDate	Date	optional	Date of the oldest transaction to appear on the statement
options.statementDate	Date	optional	Statement date
options.openTransactionsOnly	boolean true false	optional	Include only open transactions
options.inCustLocale	boolean true false	optional	Applies when advanced templates are used. Print the document in the customer's locale. If basic printing is used, this parameter is ignored and the transaction form is printed in the customer's locale.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
var transactionFile = render.statement({
  entityId: 23,
  printMode: render.PrintMode.HTML,
  inCustLocale: true
});
...
//Add additional code
```

render.transaction(options)

Method Description	Use this method to create a PDF or HTML object of a transaction.
---------------------------	--

	 Note: File size is limited to 10MB. If the Advanced PDF/HTML Templates feature is enabled, you can associate an advanced template with the custom form saved for a transaction. The advanced template is used to format the printed transaction. For details about this feature, see the help topic Advanced PDF/HTML Templates
Returns	<code>file.File</code> that contains a PDF or HTML document
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description
<code>options.entityId</code>	number	required	The internal ID of the transaction to print
<code>options.printMode</code>	enum	optional	The print output type. Set using the render.PrintMode enum. By default, uses the company/user preference for print output.
<code>options.formId</code>	number	optional	The transaction form number
<code>options.inCustLocale</code>	boolean true false	optional	Applies when advanced templates are used. Print the document in the customer's locale. If basic printing is used, this parameter is ignored and the transaction form is printed in the customer's locale.

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/render Module Script Sample .

```
//Add additional code
...
var transactionFile = render.transaction({
    entityId: 23,
    printMode: render.PrintMode.HTML,
    inCustLocale: true
});
...
//Add additional code
```

render.xmlToPdf(options)

Method Description	Method used to pass XML to the Big Faceless Organization (BFO) tag library (which is stored by NetSuite), and return a PDF file. BFO version 1.1.61 is supported in NetSuite.
---------------------------	---

	 Note: File size cannot exceed 10MB.
Returns	file.File
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/render Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description
options.xmlString	xml.Document string	required	XML document or string to convert to PDF. For information and examples about using BFO tags to create this document or string, see the help topic nlapiXMLToPDF(xmlstring) , which documents the corresponding SuiteScript 1.0 API.

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/render Module Script Sample .

```
//Add additional code
...
var pdfFile = render.xmlToPdf({
    xmlString: xmlStr
});
...
//Add additional code
```

render.DataSource

Enum Description	Holds the string values for supported data source types. Use this enum to set the <code>options.format</code> parameter. <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module

Values

- JSON
- OBJECT
- XML_DOC
- XML_STRING

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
renderer.addCustomDataSource({
  format: render.DataSource.JSON,
  alias: "JSON_STR",
  data: jsonString
});
...
//Add additional code
```

render.PrintMode

Enum Description	Holds the string values for supported print output types. Use this enum to set the <code>options.printMode</code> parameter. <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/render Module

Values

- DEFAULT
- HTML
- PDF

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/render Module Script Sample](#).

```
//Add additional code
...
```

```
printMode: render.PrintMode.HTML
...
//Add additional code
```

N/runtime Module

Load the runtime module when you want to access the current runtime settings for the script and script deployment, the user currently executing the script, and user-defined sessions.

- [N/runtime Module Members](#)
- [Script Object Members](#)
- [Session Object Members](#)
- [User Object Members](#)
- [N/runtime Module Script Sample](#)

N/runtime Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	runtime.Script	Object	Client and server-side scripts	Encapsulates the runtime settings of the currently executing script.
	runtime.Session	Object	Client and server-side scripts	Encapsulates the user session for the currently executing script.
	runtime.User	Object	Client and server-side scripts	Encapsulates the properties and preferences for the user of the currently executing script.
Method	runtime.getCurrentScript()	runtime.Script	Client and server-side scripts	Returns a runtime.Script that represents the currently executing script.
	runtime.getCurrentSession()	runtime.Session	Client and server-side scripts	Returns a runtime.Session that represents the user session for the currently executing script.
	runtime.getCurrentUser()	runtime.User	Client and server-side scripts	Returns a runtime.User that represents the properties and preferences for the user of the currently executing script.
	runtime.isFeatureInEffect(options)	boolean true false	Client and server-side scripts	Use this method to determine if a particular feature is enabled in a NetSuite account.
Property	runtime.accountId	string (read-only)	Client and server-side scripts	Returns the account ID for the currently logged-in user.
	runtime.envType	runtime.EnvType	Client and server-side scripts	Returns the current environment in which the script is executing.
	runtime.executionContext	enum	Client and server-side scripts	Returns a runtime.ContextType enumeration that represents what triggered the current script.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>runtime.processorCount</code>	number (read-only)	Client and server-side scripts	The number of processors available to the current account.
	<code>runtime.queueCount</code>	number (read-only)	Client and server-side scripts	The number of scheduled script queues available to the current account.
	<code>runtime.version</code>	string (read-only)	Client and server-side scripts	Returns the version of NetSuite that the method is called in. For example, the <code>runtime.version</code> property in an account running NetSuite 2015.2 is 2015.2 .
Enum	<code>runtime.ContextType</code>	enum	Client and server-side scripts	Enumeration that holds the context information about what triggered the current script. Returned by the <code>runtime.executionContext</code> property of the N/runtime Module .
	<code>runtime.EnvType</code>	enum	Client and server-side scripts	Enumeration that holds all possible environment types that the current script can execute in.
	<code>runtime.Permission</code>	enum	Client and server-side scripts	Enumeration that holds the user permission level for a specific permission ID. Returned by the <code>User.getPermission(options)</code> method.

Script Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Script.getParameter(options)</code>	number Date string boolean	Client and server-side scripts	Returns the value of a script parameter for the currently executing script.
	<code>Script.getRemainingUsage()</code>	number	Client and server-side scripts	Returns a number value for the usage units remaining for the currently executing script.
Property	<code>Script.deploymentsId</code>	string (read-only)	Server-side scripts	Returns the deployment ID for the script deployment on the currently executing script.
	<code>Script.id</code>	string (read-only)	Client and server-side scripts	Returns the script ID for the currently executing script.
	<code>Script.logLevel</code>	string (read-only)	Client and server-side scripts	Returns the script logging level for the current script execution. This method is not supported on client scripts.
	<code>Script.percentComplete</code>	number	Client and server-side scripts	Get or set the percent complete specified for the current scheduled script execution. The return value will appear in the %

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				Complete column on the Scheduled Script Status page.
	<code>Script.bundleIds</code>	Array (read-only)	Client and server-side scripts	Returns an Array of bundle IDs for the bundles that include the currently executing script.

Session Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Session.get(options)</code>	string	Client and server-side scripts	Returns the user-defined session object value associated with the session object key.
	<code>Session.set(options)</code>	void	Client and server-side scripts	Sets a key and value for a user-defined session object.

User Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>User.getPermission(options)</code>	number	Client and server-side scripts	Returns a user permission level for the specified permission as a runtime.Permission enumeration.
	<code>User.getPreference(options)</code>	string	Client and server-side scripts	Returns the value of a NetSuite preference. Currently only General Preferences and Accounting Preferences are exposed in SuiteScript. For more information about these preferences names and IDs, see the help topics General Preferences and Accounting Preferences .
Property	<code>User.department</code>	number (read-only)	Client and server-side scripts	Returns the internal ID of the department for the currently logged-in user.
	<code>User.email</code>	string (read-only)	Client and server-side scripts	Returns the email address of the currently logged-in user.
	<code>User.id</code>	number (read-only)	Client and server-side scripts	Returns the internal ID of the currently logged-in user.
	<code>User.location</code>	number (read-only)	Client and server-side scripts	Returns the internal ID of the location of the currently logged-in user.
	<code>User.name</code>	string (read-only)	Client and server-side scripts	Returns the name of the currently logged-in user.
	<code>User.role</code>	number (read-only)	Client and server-side scripts	Return the internal ID of the role for the currently logged-in user.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	User.roleCenter	string (read-only)	Client and server-side scripts	Returns the internal ID of the center type, or role center, for the currently logged-in user.
	User.roleId	string (read-only)	Client and server-side scripts	Returns the custom scriptId of the role for the currently logged-in user.
	User.subsidiary	number (read-only)	Client and server-side scripts	Returns the internal ID of the subsidiary for the currently logged-in user.

N/runtime Module Script Sample

Note: These samples use the `define` function. The NetSuite Debugger cannot step through a `define` function. If you need to step through your code in the NetSuite Debugger, you must use a `require` function.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following Suitelet sample writes user and session information for the currently executing script to the response.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/runtime'],
function(runtime) {
    function onRequest(context) {
        var remainingUsage = runtime.getCurrentScript().getRemainingUsage();
        var userRole = runtime.getCurrentUser().role;
        runtime.getCurrentSession().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
    };
});
```

The following scheduled script creates sales records during runtime and logs the record creation progress.

```
/**
```

```

* @NApiVersion 2.0
* @NScriptType scheduledscript
*/
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 + "%"
                });
            }
        }
    };
});

```

runtime.Script

Object Description	Encapsulates the runtime settings of the currently executing script. Use runtime.getCurrentScript() to return this object. For a complete list of this object's methods and properties, see Script Object Members .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax


Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```

//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug('Script ID: ' + scriptObj.id);
...
//Add additional code

```

Script.getParameter(options)

Method Description	Returns the value of a script parameter for the currently executing script.
Returns	number Date string Array

Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	<ul style="list-style-type: none"> The name of the script parameter. 	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug("Script parameter of custscript1: " +
    scriptObj.getParameter({name: 'custscript1'}));
...
//Add additional code
```

Script.getRemainingUsage()

Method Description	Returns a number value for the usage units remaining for the currently executing script.
Returns	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug("Remaining governance units: " + scriptObj.getRemainingUsage());
...
```

```
//Add additional code
```

Script.deploymentId

Property Description	Returns the deployment ID for the script deployment on the currently executing script.
Type	string (read-only)
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug("Deployment Id: " + scriptObj.deploymentId);
...
//Add additional code
```

Script.id

Property Description	Returns the script ID for the currently executing script.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug("Script Id: " + scriptObj.id);
...
//Add additional code
```

Script.logLevel

Property Description	Returns the script logging level for the current script execution. This method is not supported on client scripts. Returns one of the following values: <ul style="list-style-type: none"> ■ DEBUG ■ AUDIT ■ ERROR ■ EMERGENCY
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
log.debug("Logging level: " + scriptObj.logLevel);
...
//Add additional code
```

Script.percentComplete

Property Description	Get or set the percent complete specified for the current scheduled script execution. The return value appears in the % Complete column on the Scheduled Script Status page. <div>  Important: This property throws SSS_OPERATION_UNAVAILABLE if the currently executing script is not a scheduled script. </div>
Type	number
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax

 **Important:** The following code snippets show the syntax for this member. They are not a functional examples. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Gets the percentage of records completed
```

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
if (scriptObj.executionContext == ContextType.SCHEDULED)
{
    log.debug({
        details: "Script percent complete: " + scriptObj.percentComplete
    });
    ...
}
...
```

```
//Sets the percent complete
...
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 + "%"
    });
}
...
//Add additional code
```

Script.bundleIds

Property Description	Returns an Array of bundle IDs for the bundles that include the currently executing script.
Type	Array (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
var bundleArr = scriptObj.bundleIds;
...
//Add additional code
```


runtime.Session

Object Description	Encapsulates the user session for the currently executing script. Use this object to set and get user-defined objects for the current user session. Use the objects to track user-related session data. For example, you can gather information about the user scope, budget, or business problems. Use Session.set(options) to set session object values and then use Session.get(options) to retrieve the values. For a complete list of this object's methods, see Session Object Members .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Supported Script Types	Server-side scripts
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var sessionObj = runtime.getCurrentSession();
sessionObj.set({name: "myKey", value: "myValue"});
log.debug("Session object myKey value: " + sessionObj.get({name: "myKey"}));
...
//Add additional code
```

Session.get(options)

Method Description	Returns the user-defined session object value associated with the session object key.
Returns	string
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	String used as a key to store the runtime.Session .	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see link back to [N/runtime Module Script Sample](#).

```
//Add additional code
...
var sessionObj = runtime.getCurrentSession();
sessionObj.set({name: "myKey", value: "myValue"});
log.debug("Session object myKey value: " + sessionObj.get({name: "myKey"}));
...
//Add additional code
```

Session.set(options)

Method Description	Sets a key and value for a user-defined <code>runtime.Session</code> . Use <code>Session.get(options)</code> to retrieve the object value after you set it.
Returns	void
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	Key used to store the <code>runtime.Session</code> .	2015.2
options.value	string	Required	Value to associate with the key in the user session.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see link back to [N/runtime Module Script Sample](#).

```
//Add additional code
...
var sessionObj = runtime.getCurrentSession();
sessionObj.set({
  name: "myKey",
  value: "myValue"
});
```

```
log.debug("Session object myKey value: " + sessionObj.get({name: "myKey"}));
...
//Add additional code
```

runtime.User

Object Description	Encapsulates the properties and preferences for the user of the currently executing script. For a complete list of this object's methods and properties, see User Object Members .
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
...
//Add additional code
```

User.getPermission(options)

Method Description	Returns a user permission level for the specified permission as a runtime.Permission enumeration.
Returns	string
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	Internal ID of a permission. For a list of permission IDs, see Permission Names and IDs .	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see link back to [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("User permission of ADMI_ACCOUNTING:" +
    (userObj.getPermission('ADMI_ACCOUNTING') ==
    runtime.Permission.FULL?'FULL':userObj.getPermission('ADMI_ACCOUNTING')));
...
//Add additional code
```

User.getPreference(options)

Method Description	<p>Returns the value of a NetSuite preference. Currently only General Preferences and Accounting Preferences are exposed in SuiteScript. For more information about these preferences names and IDs, see the help topics General Preferences and Accounting Preferences. You can also view General Preferences by going to Setup > Company > General Preferences. View Accounting Preferences by going to Setup > Accounting > Accounting Preferences. If you want to change the value of a General or Accounting preference using SuiteScript 2.0, you must load each preference page using config.load(options), where <code>options.name</code> is <code>COMPANY_PREFERENCES</code> or <code>ACCOUNTING_PREFERENCES</code>. The config.load(options) method returns a record.Record. You can use the Record.setValue(options) method to set the preference.</p> <div> Note: The permission level will be Permission.FULL if the script is configured to execute as admin. You can configure a script to execute as admin by selecting "administrator" from the Execute as Role field on Script Deployment page. </div>
Returns	string
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	Internal ID of the preference. For a list of preference IDs, see <i>Preference Names and IDs</i> .	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see link back to [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("User preference for emailapproval: " + userObj.getPreference({name: "emailapproval"}));
...
//Add additional code
```

User.department

Property Description	Returns the internal ID of the department for the currently logged-in user.
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Internal ID of current user department: " + userObj.department);
...
//Add additional code
```

User.email

Property Description	<p>Returns the email address of the currently logged-in user. To use this property, the email field on the user employee record must contain an email address.</p> <div> Note: In a shopping context where the shopper is recognized but not logged in, this method can be used to return the shopper's email, instead of getting it from the customer record. </div>
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Current user email: " + userObj.email);
...
//Add additional code
```

User.id

Property Description	Returns the internal ID of the currently logged-in user.
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Internal ID of current user: " + userObj.id);
...
//Add additional code
```

User.location

Property Description	Returns the internal ID of the location of the currently logged-in user.
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module

Since	2015.2
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Internal ID of current user location: " + userObj.location);
...
//Add additional code
```

User.name

Property Description	Returns the name of the currently logged-in user.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Name of current user: " + userObj.name);
...
//Add additional code
```

User.role

Property Description	Return the internal ID of the role for the currently logged-in user.
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Internal ID of current user role: " + userObj.role);
...
//Add additional code
```

User.roleCenter

Property Description	Returns the string value of the center type, or role center, for the currently logged-in user. The NetSuite user interface adjusts automatically to different users' business needs. For each user, NetSuite displays a variable set of tabbed pages, called a center , based on the user's assigned role. Each NetSuite center provides, for users with related roles, the pages and links they need to do their jobs. For more information about NetSuite centers, see the help topic Centers Overview .
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("String value of current user center type (role center): " + userObj.roleCenter);
...
//Add additional code
```

User.roleId

Property Description	Returns the custom scriptId of the role for the currently logged-in user. You can use this value instead of the internal ID for the role. When bundling a custom role, the internal ID number of the role in the target account can change after the bundle is installed. Therefore, in the target account you can use this property to return the unique/ custom scriptId assigned to the role.
Type	string
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Custom script ID of current user role: " + userObj.roleId);
...
//Add additional code
```

User.subsidiary

Property Description	Returns the internal ID of the subsidiary for the currently logged-in user.
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
log.debug("Internal ID of current user subsidiary: " + userObj.subsidiary);
...
//Add additional code
```

runtime.getCurrentScript()

Method Description	Returns a runtime.Script that represents the currently executing script. Use this method to get properties and parameters of the currently executing script and script deployment. If you want to get properties for the session or user, use runtime.getCurrentSession() or runtime.getCurrentUser() instead.
Returns	runtime.Script
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var scriptObj = runtime.getCurrentScript();
...
//Add additional code
```

runtime.getCurrentSession()

Method Description	Returns a runtime.Session that represents the user session for the currently executing script. Use this method to get session objects for the current user session. If you want to get properties for the script or user, use runtime.getCurrentScript() or runtime.getCurrentUser() instead.
Returns	runtime.Session
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var sessionObj = runtime.getCurrentSession();
...
//Add additional code
```

runtime.getCurrentUser()

Method Description	Returns a runtime.User that represents the properties and preferences for the user of the currently executing script. Use this method to get session objects for the current user session. If you want to get properties for the script or session, use runtime.getCurrentScript() or runtime.getCurrentSession() instead.
---------------------------	---

Returns	<code>runtime.User</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
var userObj = runtime.getCurrentUser();
...
//Add additional code
```

runtime.isFeatureInEffect(options)

Method Description	Use this method to determine if a particular feature is enabled in a NetSuite account. These are the features that appear on the Enable Features page at Setup > Company > Enable Features.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/runtime Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.feature</code>	string	Required	The internal ID of the feature to check. For a list of feature internal IDs, see the help topic Feature Names and IDs .	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
```

```
...
log.debug('Advanced Billing feature is enabled: ' + runtime.isFeatureInEffect({feature: "ADVBILLING"}));
...
//Add additional code
```

runtime.accountId

Property Description	Returns the account ID for the currently logged-in user.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
log.debug("Account ID for the current user: " + runtime.accountId);
...
//Add additional code
```

runtime.envType

Property Description	Returns the current environment in which the script is executing. This property returns one of the values from the runtime.EnvType enumeration.
Type	runtime.EnvType
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
log.debug("Environment for current user: " + JSON.stringify(runtime.envType));
```

```
...
//Add additional code
```

runtime.executionContext

Property Description	Property that describes what triggered the current script. This value is set by the runtime.ContextType enumeration.
Type	runtime.ContextType
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
if (runtime.executionContext === runtime.ContextType.USEREVENT)
    return;
...
//Add additional code
```

runtime.processorCount

Property Description	<p>The number of processors available to the currently logged in account. SuiteCloud Processors is the current system used to execute (process) scheduled scripts and map/reduce scripts. This property is helpful if you are a SuiteApp developer and your script needs to know the total number of processors available to a deployment.</p> <p>For scheduled script deployments that continue to use queues, use runtime.queueCount. With the introduction of SuiteCloud Processors, map/reduce script deployments and new scheduled script deployments no longer use queues, but pre-existing scheduled script deployments continue to use queues until the queues are removed (see the help topic SuiteCloud Processors – Supported Task Types).</p> <p>Be aware that the number of processors available may not be the same as the number of queues available. For more information, see the help topic SuiteCloud Plus Settings.</p> <div data-bbox="402 1633 1370 1780"> <p> Note: The <code>runtime.processorCount</code> property reflects the number of processors available to an account. It is not impacted by changes to deployments. The value is the same regardless of whether deployments continue to use queues. For more information, see the help topic SuiteCloud Processors – Supported Task Types.</p> </div> <p>For more information on scheduled scripts, see SuiteScript 2.0 Scheduled Script Type. For more information on map/reduce scripts, see SuiteScript 2.0 Map/Reduce Script Type.</p>
Type	number (read-only)

Module	N/runtime Module
Since	2018.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
log.debug("Number of processors available: " + runtime.processorCount);
...
//Add additional code
```

runtime.queueCount

Property Description	<p>The number of queues available to the currently logged in account.</p> <p>SuiteCloud Processors is the current system used to execute (process) scheduled scripts and map/reduce scripts. This property is helpful if you are a SuiteApp developer and your script needs to know the total number of queues available to a deployment.</p> <p>For map/reduce script deployments, use runtime.processorCount. With the introduction of SuiteCloud Processors, no map/reduce script deployments use queues (see the help topic SuiteCloud Processors – Supported Task Types).</p> <p>Be aware that the number of queues available may not be the same as the number of processors available (see the help topic SuiteCloud Plus Settings).</p> <div data-bbox="406 1092 1370 1239"> <p>Note: If all scheduled script deployments in an account are configured to no longer use queues (see the help topic SuiteCloud Processors – Supported Task Types), the value of <code>runtime.queueCount</code> is unchanged. This property reflects the number of queues available to an account. It is not impacted by changes to deployments.</p> </div> <p>For more information on scheduled scripts, see SuiteScript 2.0 Scheduled Script Type.</p>
Type	number (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
log.debug("Number of queues available: " + runtime.queueCount);
...
//Add additional code
```

runtime.version

Property Description	Returns the version of NetSuite that the method is called in. For example, the runtime.version property in an account running NetSuite 2015.2 is 2015.2 . Use this method, for example, when installing a bundle in another NetSuite accounts and you want to know the version number before installing the bundle.
Type	string (read-only)
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/runtime Module Script Sample](#).

```
//Add additional code
...
log.debug("Current NetSuite version: " + runtime.version);
...
//Add additional code
```

runtime.ContextType

Enum Description	Enumeration used to set the runtime.executionContext property. The runtime.executionContext property describes what triggered the current script. <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Values

Enum Value	Sets runtime.ExecutionContext Property To
ACTION	ACTION
BUNDLE_INSTALLATION	BUNDLE_INSTALLATION
CLIENT	CLIENT
CONSOLRATEADJUSTOR	CONSOLRATEADJUSTOR

Enum Value	Sets runtime.ExecutionContext Property To
CSV_IMPORT	CSVIMPORT
CUSTOMGLLINES	CUSTOMGLLINES
CUSTOM_MASSUPDATE	CUSTOMMASSUPDATE
DEBUGGER	DEBUGGER
EMAIL_CAPTURE	EMAILCAPTURE
MAP_REDUCE	MAPREDUCE
PAYMENTGATEWAY	PAYMENTGATEWAY
PORTLET	PORTLET
PROMOTIONS	PROMOTIONS
RESTLET	RESTLET
SCHEDULED	SCHEDULED
SHIPPING_PARTNERS	SHIPPINGPARTNERS
SUITELET	SUITELET
TAX_CALCULATION	TAXCALCULATION
USEREVENT	USEREVENT
USER_INTERFACE	USERINTERFACE
WEBAPPLICATION	WEBAPPLICATION
WEBSERVICES	WEBSERVICES
WEBSTORE	WEBSTORE
WORKFLOW	WORKFLOW

runtime.EnvType

Enum Description	<p>Enumeration that holds all possible environment types that the current script can execute in. One of these values is returned by the runtime.envType property.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/runtime Module
Since	2015.2

Values

<div> <div></div> <div>SANDBOX</div> </div>

- PRODUCTION
- BETA
- INTERNAL

runtime.Permission

Enum Description	<p>Enumeration that holds the user permission level for a specific permission ID. Returned by the User.getPermission(options) method. See the help topic Permission Names and IDs. For information on working with NetSuite permissions, see the help topic NetSuite Permissions Overview.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Client and server-side scripts For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/runtime Module
Since	2015.2

Values

- FULL
- EDIT
- CREATE
- VIEW
- NONE

N/search Module

Load the search module to create and run on demand or saved searches and analyze and iterate through the search results. You can search for a single record by keywords, create saved searches, search for duplicate records, or return a set of records that match filters you define.

You also have the option to paginate search results and construct navigation that jumps between the next and previous pages. Due to the performance benefits, this is a suitable approach for working with a large result set.

- [N/search Module Members](#)
- [Search Object Members](#)
- [Result Object Members](#)
- [Column Object Members](#)
- [Filter Object Members](#)
- [Page Object Members](#)
- [PagedData Object Members](#)
- [PageRange Object Members](#)

- [ResultSet Object Members](#)
- [N/search Module Script Samples](#)

N/search Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	search.Search	Object	Client and server-side scripts	Encapsulates a NetSuite search. Use the methods available to the Search object to create a search, run a search, or save a search.
	search.Result	Object	Client and server-side scripts	Encapsulate a single search result row. Use the methods and properties for the Result object to get the column values for the result row.
	search.Column	Object	Client and server-side scripts	Encapsulates a single search column in a search.Search object. Use the methods and properties available to the Column object to get or set Column properties.
	search.Filter	Object	Client and server-side scripts	Encapsulates a search filter used in a search. Use the properties for the Filter object to get and set the filter properties.
	search.ResultSet	Object	Client and server-side scripts	Encapsulates a set of search results returned by Search.run() .
	search.Page	Object	Client and server-side scripts	Encapsulates a set of search results for a single search page.
	search.PagedData	Object	Client and server-side scripts	Holds metadata about a paginated query.
	search.PageRange	Object	Client and server-side scripts	Defines the page range to bound the result set for a paginated query.
	search.Setting	Object	Client and server-side scripts	Encapsulates a search setting. Search settings let you specify search parameters that are typically available only in the UI.
Method	search.create(options)	search.Search	Client and server-side scripts	Creates a new search and returns it as a search.Search object.
	search.create.promise(options)	search.Search	Client scripts	Creates a new search asynchronously and returns it as a search.Search object.
	search.load(options)	search.Search	Client and server-side scripts	Loads an existing saved search and returns it as a search.Search object.
	search.load.promise(options)	search.Search	Client scripts	Loads an existing saved search asynchronously and returns it as a search.Search object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	search.delete(options)	void	Client and server-side scripts	Deletes an existing saved search asynchronously and returns it as a search.Search object.
	search.delete.promise(options)	void	Client scripts	Deletes an existing saved search and returns it as a search.Search object.
	search.duplicates(options)	search.Result[]	Client and server-side scripts	Performs a search for duplicate records based on the duplicate detection configuration for the account. Returns an array of search.Result objects.
	search.duplicates.promise(options)	search.Result[]	Client scripts	Performs a search for duplicate records asynchronously based on the duplicate detection configuration for the account. Returns an array of search.Result objects.
	search.global(options)	search.Result[]	Client and server-side scripts	Performs a global search against a single keyword or multiple keywords.
	search.global.promise(options)	search.Result[]	Client scripts	Performs a global search asynchronously against a single keyword or multiple keywords.
	search.lookupFields(options)	Object array	Client and server-side scripts	Performs a search for one or more body fields on a record. Returns select fields as an object with value and text properties. Returns multiselect fields as an object with value:text pairs.
	search.lookupFields.promise(options)	Object array	Client scripts	Performs a search asynchronously for one or more body fields on a record. Returns select fields as an object with value and text properties. Returns multiselect fields as an object with value:text pairs.
	search.createColumn(options)	search.Column	Client and server-side scripts	Creates a new search column as a search.Column object.
	search.createFilter(options)	search.Filter	Client and server-side scripts	Creates a new search filter as a search.Filter object.
	search.createSetting(options)	search.Setting	Client and server-side scripts	Creates a new search setting and returns it as a search.Setting object.
Enum	search.Operator	enum	Client and server-side scripts	Enumeration that holds the values for search operators to use with the search.Filter object.
	search.Sort	enum	Client and server-side scripts	Enumeration that holds the values for supported sorting directions used with search.createColumn(options) .

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	search.Summary	enum	Client and server-side scripts	Enumeration that holds the values for summary types used by the Column.summary object.
	search.Type	enum	Client and server-side scripts	Enumeration that holds the string values for search types supported in the N/search Module . This enum is used to pass the type argument to search.create(options) .

Search Object Members

The following members are called on [search.Search](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Search.save()	number	Client and server-side scripts	Saves a search created by search.create(options) or loaded with search.load(options) . Returns the internal ID of the saved search.
	Search.save.promise()	number	Client scripts	Asynchronously saves a search created by search.create(options) or loaded with search.load(options) . Returns the internal ID of the saved search.
	Search.run()	search.ResultSet	Client and server-side scripts	Runs an on demand search created with search.create(options) or a search loaded with search.load(options) , returning the results as a search.ResultSet .
	Search.runPaged(options)	search.PagedData	Client and server-side scripts	Runs the current search and returns a search.PagedData Object.
	Search.runPaged.promise(options)	search.PagedData	Client scripts	Asynchronously runs the current search and returns a search.PagedData Object.
Property	Search.searchType	string	Client and server-side scripts	Search type on which a search is based.
	Search.searchId	number	Client and server-side scripts	Internal ID of a search.
	Search.filters	search.Filter[]	Client and server-side scripts	Filters for the search as an array of search.Filter objects.
	Search.filterExpression	Object[]	Client and server-side scripts	Search filter expression for the search as an array of expression objects.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Search.columns	search.Column[] string[]	Client and server-side scripts	Columns to return for this search as an array of search.Column objects or a string array of column names.
	Search.settings	search.Setting[]	Client and server-side scripts	Search settings for this search as an array of search.Setting objects.
	Search.title	string	Client and server-side scripts	Title for a saved search. Use this property to set the title for a search before you save it for the first time.
	Search.id	string	Client and server-side scripts	Script ID for a saved search, starting with <code>customsearch</code> .
	Search.isPublic	boolean <code>true</code> <code>false</code>	Client and server-side scripts	Value is <code>true</code> if the search is public, or <code>false</code> if it is not.

Column Object Members

The following members are called on [search.Column](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Column.setWhenOrderedBy(options)	search.Column	Client and server-side scripts	Returns the search column for which the minimal or maximal value should be found when returning the search.Column value.
Property	Column.name	string (read-only)	Client and server-side scripts	Name of a search column as a string.
	Column.join	string (read-only)	Client and server-side scripts	Join ID for a search column as a string.
	Column.summary	string (read-only)	Client and server-side scripts	Returns the summary type for a search column.
	Column.formula	string	Client and server-side scripts	Formula used for a search column as a string.
	Column.label	string	Client and server-side scripts	Label used for the search column. You can only get or set custom labels with this property.
	Column.function	string	Client and server-side scripts	Special function used in the search column as a string.

Filter Object Members

The following members are called on [search.Filter](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Filter.name	string (read-only)	Client and server-side scripts	Name or internal ID of the search field.
	Filter.join	string (read-only)	Client and server-side scripts	Join ID for the search filter.
	Filter.operator	string (read-only)	Client and server-side scripts	Operator used for the search filter.
	Filter.summary	search.Summary	Client and server-side scripts	Summary type for the search filter.
	Filter.formula	string	Client and server-side scripts	Formula used by the search filter.

Page Object Members

The following members are called on the [search.Page](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Page.next()	void	Client and server-side scripts	Gets the next segment of data from a paginated search
	Page.next.promise()	void	Client scripts	Asynchronously gets the next segment of data from a paginated search
	Page.prev()	void	Client and server-side scripts	Gets the previous segment of data from a paginated search
	Page.prev.promise()	void	Client scripts	Asynchronously gets the previous segment of data from a paginated search
Property	Page.data	search.Result[]	Client and server-side scripts	The results from a paginated search.
	Page.isFirst	read-only boolean	Client and server-side scripts	Indicates whether a page is the first page of data for a result set
	Page.isLast	read-only boolean	Client and server-side scripts	Indicates whether a page is the last page of data for a result set.
	Page.pagedData	read-only search.PagedData	Client and server-side scripts	The PagedData Object used to fetch this Page Object.
	Page.pageRange	read-only search.PageRange	Client and server-side scripts	The PageRange Object used to fetch this Page Object.

PagedData Object Members

The following members are called on [search.PagedData](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	PagedData.fetch(options)	search.Page	Client and server-side scripts	Retrieves the data within the specified page range.
	PagedData.fetch.promise()	search.Page	Client scripts	Asynchronously retrieves the data within the specified page range.
Property	PagedData.count	read-only number	Client and server-side scripts	The total number of results when Search.runPaged(options) was executed.
	PagedData.pageRanges	read-only search.PageRange[]	Client and server-side scripts	The collection of PageRange objects that divide the entire result set into smaller groups.
	PagedData.pageSize	read-only number	Client and server-side scripts	The maximum number of entries per page
	PagedData.searchDefinition	read-only search.Search	Client and server-side scripts	The search criteria used when Search.runPaged(options) was executed.

PageRange Object Members

The following members are called on [search.PageRange](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	PageRange.compoundLabel	read-only string	Client and server-side scripts	Human-readable label with beginning and ending range identifiers
	PageRange.index	read-only number	Client and server-side scripts	The index of this page range.

Result Object Members

The following members are called on [search.Result](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Result.getValue(options)	string	Client and server-side scripts	Used on formula fields and non-formula (standard) fields to get the value of a specified search return column.
	Result.getValue(column)	string	Client and server-side scripts	Used on formula and non-formula (standard) fields. Returns the string value of a specified search result column. For convenience, this method takes a single search.Column Object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>Result.getText(column)</code>	string	Client and server-side scripts	The text value for a search.Column if it is a stored select field.
	<code>Result.getText(options)</code>	string	Client and server-side scripts	The UI display name, or text value, for a search result column. This method is supported only for non-stored select, image, and document fields.
Property	<code>Result.recordType</code>	string (read-only)	Client and server-side scripts	The type of record returned in a search result row.
	<code>Result.id</code>	number(read-only)	Client and server-side scripts	The internal ID for the record returned in a search result row.
	<code>Result.columns</code>	search.Column []	Client and server-side scripts	Array of search.Column objects that encapsulate the columns returned in the search result row.

ResultSet Object Members

The following members are called on [search.ResultSet](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>ResultSet.getRange(options)</code>	search.Result []	Client and server-side scripts	Retrieve a slice of the search result as an array of search.Result objects.
	<code>ResultSet.getRange.promise(options)</code>	search.Result []	Client scripts	Asynchronously retrieve a slice of the search result as an array of search.Result objects.
	<code>ResultSet.each(callback)</code>	void	Client and server-side scripts	Use a developer-defined function to invoke on each row in the search results, up to 4000 results at a time.
	<code>ResultSet.each.promise(callback)</code>	void	Client scripts	Asynchronously use a developer-defined function to invoke on each row in the search results, up to 4000 results at a time.
Property	<code>ResultSet.columns</code>	search.Column []	Client and server-side scripts	An array of search.Column objects that represent the columns returned in the search results.

Setting Object Members

The following members are called on [search.Setting](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	Setting.name	read-only string	Client and server-side scripts	The name of the search parameter.
	Setting.value	read-only string	Client and server-side scripts	The value of the search parameter.

N/search Module Script Samples

In your NetSuite account, the One World feature needs to be enabled in the account for the samples to work. These samples are designed to run from a OneWorld account.

Note: These sample scripts use the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following examples create a saved search on the sales order record.

```
/**
 *@NApiVersion 2.x
 */
// This example creates a saved search on the salesorder record
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();
});
```

```
/**
 *@NApiVersion 2.x
 */
require(['N/search'],
function(search){
    function createAnotherSearch() {
        var mySalesOrderSearch = search.create({
            type: search.Type.SALES_ORDER,
            title: 'My Second SalesOrder Search',
            id: 'customsearch_my_second_so_search',
```

```

        columns: [{
            name: 'entity'
        }, {
            name: 'subsidiary'
        }, {
            name: 'name'
        }, {
            name: 'currency'
        }],
        filters: [{
            name: 'mainline',
            operator: 'is',
            values: ['T']
        }]
    });
    mySalesOrderSearch.save();
}
createAnotherSearch();
});

```

The following example loads and runs a search on the sales order record, and uses a callback function on the results.

```

/**
 *@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();
});

```

The following example loads and runs a search on the sales order record, and gets the first 100 rows of results.

```

/**
 *@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();
});

```

The following example loads and runs a search on the sales order record, and uses a callback function on the paginated results.

```

/**
 *@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();
    });

```

The following example deletes a saved search.

```

/**
 *@NApiVersion 2.x
 */
require(['N/search'],
    function(search) {
        function deleteSearch() {
            search.delete({

```

```

        id: 'customsearch_my_so_search'
    });
}
deleteSearch();
});

```

search.Search

Object Description	<p>Encapsulates a NetSuite search. Use the methods available to search.Search to create a search, run a search, or save a search.</p> <div>  Note: You do not need to save the search to run it. </div> <p>For more information about executing NetSuite searches using SuiteScript, see <i>Searching Overview</i>. For a complete list of this object's methods and properties, see Search Object Members.</p>
Supported Script Types	<p>All script types For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/search Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});
...
//Add additional code

```

Search.run()

Method Description	<p>Runs an on demand search created with search.create(options) or a search loaded with search.load(options), returning the results as a search.ResultSet. Calling this method does not save the search.</p> <p>Use this method with search.create(options) to create and run on demand searches that are never saved to the database.</p> <p>After you run a search, you can use ResultSet.each(callback) to iterate through the result set and process each result.</p> <div>  Important: Search result sets are not cached. If records applicable to your search are created, modified, or deleted at the same time you are traversing your result set, your result set may change. </div>
Returns	search.ResultSet

Governance	None
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
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;
    });
}
```

Search.runPaged(options)

Method Description	<p>Runs the current search and returns summary information about paginated results. Calling this method does not give you the result set or save the search. To retrieve data, use PagedData.fetch(options).</p> <div>  Important: When you use this method to run a paged search, consider the following: <ul style="list-style-type: none"> ■ Search result sets are not cached. If records applicable to your search are created, modified, or deleted at the same time you are traversing your result set, your result set may change. ■ This method can return a maximum of 1000 pages of search results. </div>
Returns	search.PagedData
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.pageSize	number	optional	<p>Maximum number of entries per page There is an upper limit, a lower limit, and a default setting:</p> <ul style="list-style-type: none"> ■ The maximum number allowed is 1000. ■ The minimum number allowed is 5. ■ By default, the page size is set to 50 entries per page.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
// Run the paged search
var pagedData = mySearch.runPaged({
    pageSize: 50
});
...
// Use the count property to count the
// search results easily
var resultCount = mySearch.runPaged({
    pageSize: 50
}).count;
...
//Add additional code
```

Search.runPaged.promise(options)

Method Description	<p>Runs the current search asynchronously and returns a search.PagedData Object.</p> <div>  Note: For more information about using this method, see Search.runPaged(options). For additional information on promises, see Promise Object. </div>
Returns	search.PagedData
Synchronous Version	Search.runPaged(options)
Supported Script Types	<p>All client-side scripts For more information, see SuiteScript 2.0 Client Script Type.</p>
Governance	5 units
Module	N/search Module

Since	2016.1
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
mySearch.runPaged.promise().then(getPageRangesPromiseChain);
...
//Add additional code
```

Search.save()

Method Description	<p>Saves a search created by search.create(options) or loaded with search.load(options). Returns the internal ID of the saved search.</p> <p>You must set the title and id properties for a new saved search before you save it, either when you create it with search.create(options) or by setting the Search.title and Search.id properties. If you do not set the saved search ID, NetSuite generates one for you. See Search.id.</p> <div data-bbox="422 924 1372 1018"> <p> Note: You do not need to set these properties if you load a previously saved search with search.load(options) and then save it.</p> </div> <p>This method also includes a promise version, Search.save.promise(). For more information about promises, see Promise Object.</p>
Returns	the internal search ID of the saved search as a number
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required Search.title property not set on search.Search .
NAME_ALREADY_IN_USE	A search has already been saved with that name. Please use a different name.	The Search.title property on search.Search is not unique.
SSS_DUPLICATE_SEARCH_SCRIPT_ID	Saved search script IDs must be unique. Please choose another script ID. If you are trying to modify an existing saved search, use search.load() .	The Search.id property on search.Search is not unique.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
mySalesOrderSearch.save();
...
//Add additional code
```

Search.save.promise()

Method Description	Asynchronously saves a search created by search.create(options) or loaded with search.load(options) . Returns the internal ID of the saved search.
	<div> Note: For more information about using this method, see Search.save(). For additional information on promises, see Promise Object. </div>
Returns	number
Synchronous Version	Search.save()
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	5 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.create.promise({
    type: search.Type.SALES_ORDER
})
.then(function(searchObj) {
    return searchObj.save.promise()
})
.then(function (result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function onRejected(reason) {
```



```

        // do something on rejection
    });
    ...
    //Add additional code

```

Search.searchType

Property Description	Internal ID name of the record type on which a search is based. Use this if you have the internal ID of the search, but do not know the record type the search was based on. For example, if the search was on a Customer record, this property is <code>customer</code> ; if the search was on the Sales Order record type, this property is <code>salesorder</code> .
Type	read-only string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});
log.debug({
    title: 'record type: ',
    details: mySearch.searchType
});
...
//Add additional code

```

Search.searchId

Property Description	Internal ID of the search. The internal ID is available only when the search is either loaded with <code>search.load(options)</code> or after it has been saved with <code>Search.save()</code> . Typical values are 55 or 234 or 87, not a value like <code>customsearch_mysearch</code> . Any ID prefixed with <code>customsearch</code> is a script ID, not the internal system ID for a search.
Type	number
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.load({
  id: 'customsearch_my_so_search'
});
log.debug({
  title: 'search id #: ',
  details: mySearch.searchId
});
...
//Add additional code
```

Search.filters

Property Description	<p>Filters for the search as an array of search.Filter objects. Value is <code>null</code> if the search has no defined filters.</p> <p>You set this value with an array or single search.Filter objects to overwrite any prior filters. Use <code>null</code> to set an empty array and remove any existing filters on this search. Use search.createFilter(options) to create a filter.</p> <div>  Note: If you want to get or set a search filter expression, use the Search.filterExpression property. </div>
Type	search.Filter []
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_INVALID_SRCH_FILTER	An search filter contains invalid search criteria	Invalid value for search filter type.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...

var myFilter = search.createFilter({
  name: 'entity',
  operator: search.Operator.ISEMPY,
```

```
});

function createSearch() {
    var mySalesOrderSearch = search.create({
        type: search.Type.SALES_ORDER,
        filters: myFilter
    });
    ...
    //Add additional code
}
```

Search.filterExpression

Property Description	<p>Use filter expressions as a shortcut to create filters (search.Filter). A search filter expression is a JavaScript string array of zero or more elements. Each element is one of the following:</p> <ul style="list-style-type: none"> Operator - For a list of supported operators, see search.Operator. Filter term Two or more filter expressions combined logically with 'and', 'or', or 'not' <p>Use <code>null</code> to set an empty array and remove any existing filter expressions on this search.</p> <div>  Note: If you want to get or set search filters, use the Search.filters property. </div>
Type	Object[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_INVALID_SRCH_FILTER_EXPR	<p>Malformed search filter expression. This is a general error raised when a filter expression cannot be parsed. For example:</p> <pre>[f1, 'and', 'and', f2]</pre>	The <code>options.filters</code> parameter is not a valid search filter, filter array, or filter expression.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
Search.create({
    type: search.Type.CUSTOMER,
    filters: [
        ['email', search.Operator.STARTSWITH, 'kwolff'], 'and', [
            ['id', search.Operator.EQUALTO, 107], 'or',
            ['id', search.Operator.EQUALTO, 2508]
        ]
    ]
});
```

```

    ]
  });

  ...
  //Add additional code

```

Search.columns

Property Description	Columns to return for this search as an array of search.Column objects or a string array of column names. You set this value with an array of search.Column objects or a single search.Column to overwrite any prior return columns for the search. Use <code>null</code> to set an empty array and remove any existing columns on this search.
Type	search.Column[] string[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_INVALID_SRCH_COLUMN		The value passed in was not a string or search.Column Object

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
function createSearch() {
    var mySalesOrderSearch = search.create({
        type: search.Type.SALES_ORDER,
        columns: ['entity', 'subsidiary', 'name', 'currency'],
    });
    ...
    //Add additional code

```

Search.settings

Property Description	Search settings for this search as an array of search.Setting objects. Search settings let you specify search parameters that are typically available only in the UI. You set this value with an array of search.Setting objects or a single search.Setting object. You can create a search.Setting object by calling search.createSetting(options) . The supported values for a search.Setting object differ depending on the search parameter that you set. For more information, see Setting.name and Setting.value .
Type	search.Setting[]

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2018.2

Errors

Error Code	Message	Thrown If
SSS_INVALID_SRCH_SETTING		An unknown search parameter name is provided.
SSS_INVALID_SRCH_SETTING_VALUE		An unsupported value is set for the provided search parameter name.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
  type: 'transaction',
  columns: [ 'trandate', 'amount', 'entity' ],
  filters: [
    search.createFilter({
      name: 'internalid',
      operator: search.Operator.ANYOF,
      values: [13, 12356]
    })
  ],
  settings: [
    search.createSetting({
      name: 'consolidationtype',
      value: 'NONE'
    })
  ]
});
...
//Add additional code
```

Search.title

Property Description	Title for a saved search. Use this property to set the title for a search before you save it for the first time. You can also set the title for a search when you create it with search.create(options) . The Search.title property is required to save a search with Search.save() .
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
function createSearch() {
    var mySalesOrderSearch = search.create({
        type: search.Type.SALES_ORDER,
        title: 'My SalesOrder Search',
        id: 'customsearch_my_so_search',
    });
    mySalesOrderSearch.save();
    ...
//Add additional code
```

Search.id

Property Description	Script ID for a saved search, starting with <code>customsearch</code> . If you do not set this property and then save the search, NetSuite generates a script ID for you.
	<div>  Note: This is not the internal NetSuite ID for the saved search. See Search.searchId. </div>
Type	number
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
function createSearch() {
    var mySalesOrderSearch = search.create({
        type: search.Type.SALES_ORDER,
        title: 'My SalesOrder Search',
        id: 'customsearch_my_so_search',
    });
    ...
//Add additional code
```

Search.isPublic

Property Description	Value is <code>true</code> if the search is public, or <code>false</code> if it is not. By default, all searches created through <code>search.create(options)</code> are private.
-----------------------------	---

Type	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});
mySearch.isPublic = true;
...
//Add additional code
```

search.Result

Object Description	<p>Encapsulate a single search result row. Use the methods and properties for <code>search.Result</code> to get the column values for the result row.</p> <div data-bbox="433 1081 1370 1142" data-label="Text"> <p> Note: Use <code>search.ResultSet</code> for the set of results from a search.</p> </div> <p>For more information about executing NetSuite searches using SuiteScript, see <i>Searching Overview</i>. For a complete list of this object's methods and properties, see Result Object Members.</p>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
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'
    });
    ...
    //Add additional code

```

Result.getValue(column)

Method Description	Used on formula and non-formula (standard) fields. Returns the string value of a specified search result column. For convenience, this method takes a single search.Column Object. <div>  Note: This method is overloaded. You can also use Result.getValue(options) to get column values based on the name, join, and summary values for a column. </div>
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description
column	search.Column	Required	The search result column from which to return a value.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});

var resultSet = mySearch.run();
var firstResult = resultSet.getRange({
    start: 0,
    end: 1
})[0];

// get the value of the second column (zero-based index)

```



```

var value = firstResult.getValue(resultSet.columns[1]);

log.debug({
    title: 'Value:',
    details: value
});

...
//Add additional code

```

Result.getValue(options)

Method Description	<p>Used on formula and non-formula (standard) fields. Returns the string value of a specified search result column. Takes in arguments for name, join, and summary.</p> <div>  Note: This method is overloaded. You can also use Result.getValue(column) to get column values. This method takes in a single search.Column. </div> <div>  Important: If you have multiple search return columns and you apply grouping, all columns must include a summary property. </div>
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description
options.name	string	Required	The search return column name.
options.join	string	Optional	The join id for this search return column. Join IDs are listed in the Records Browser. For more information, see the help topic Working with the SuiteScript Records Browser .
options.summary	search.Summary	Optional	The summary type for this column. See search.Summary .
options.func	string	Optional	Special function for the search column. See Column.function .

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/search Module Script Samples .
--

```
//Add additional code
```

```

...
var searchResults = mySearch.run().getRange({
    start: 0,
    end: 100
});
for (var i = 0; i < searchResults.length; i++) {
    var amount = searchResults[i].getValue({
        name: 'amount'
    });
    var entity = searchResults[i].getValue({
        name: 'name',
        join: 'location'
    });
}
...
//Add additional code

```

Result.getText(column)

Method Description	Used on select, image, and document fields. Returns the text value of a specified search result column. For convenience, this method takes a single search.Column Object. <div> Note: This method is overloaded. You can also use Result.getText(options) to get column text value based on the name, join and summary values for a column.</div>
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

Parameter	Type	Required / Optional	Description
column	search.Column	Required	Name of the search result column.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});

var resultSet = mySearch.run();
var firstResult = resultSet.getRange({

```

```

        start: 0,
        end: 1
    }][0];

    // get the text value of the second column (zero-based index)
    var value = firstResult.getText(resultSet.columns[1]);

    log.debug({
        title: 'Value: ',
        details: value
    });
    ...
    //Add additional code

```

Result.getText(options)

Method Description	Used on select, image, and document fields. Returns the text value of a specified search result column.
	 Note: This method is overloaded. You can also use Result.getText(column) to get a column value. This method takes in a single search.Column .
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	The name of the search column.
options.join	string	Optional	The join internal ID for the search column.
options.summary	search.Summary	Optional	The summary type used for the search column. See search.Summary .

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var searchResults = mySearch.run().getRange({

```

```

        start: 0,
        end: 100
    });
    for (var i = 0; i < searchResults.length; i++) {
        var amount = searchResults[i].getText({
            name: 'amount'
        });
        var entity = searchResults[i].getText({
            name: 'name',
            join: 'location'
        });
        ...
    }
    //Add additional code

```

Result.recordType

Property Description	The type of record returned in a search result row.
Type	search.Type enum
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});

var searchResult = mySearch.run();
log.debug({
    title: 'Record Type: ',
    details: searchResult.recordType
});
...
//Add additional code

```

Result.id

Property Description	The internal ID for the record returned in a search result row.
Type	number (read-only)
Supported Script Types	All script types

	For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.load({
  id: 'customsearch_my_so_search'
});

var resultSet = mySearch.run();
resultSet.each(function(result) {
  log.debug({
    title: 'Record Internal ID: ',
    details: result.id
  });
  return true;
});

...
//Add additional code
```

Result.columns

Property Description	Array of search.Column objects that encapsulate the columns returned in the search result row.
Type	search.Column[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.load({
  id: 'customsearch_my_so_search'
});
```

```

var firstResult = mySearch.run().getRange({
    start: 0,
    end: 1
})[0];
log.debug({
    details: "There are " + firstResult.columns.length + " columns in the result."
});

firstResult.columns.forEach(function(col) { // log each column
    log.debug({
        details: col
    });
});
...
//Add additional code

```

search.Column

Object Description	<p>Encapsulates a single search column in a search.Search. Use the methods and properties available to the Column object to get or set Column properties.</p> <p>You create a search column object with search.createColumn(options) and add it to a search.Search object that you create with search.create(options) or load with search.load(options).</p> <p>You can pass a Column object as a parameter to the Result.getValue(column) or Result.getText(column) methods.</p> <p>In addition, search.ResultSet contains an array of Column objects returned in the results of a search.</p> <p>For a complete list of this object's methods and properties, see Column Object Members.</p>
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
search.create({
    type: search.Type.TRANSACTION,
    columns: [
        'trandate',
        'amount',
        'entity',
        'entity.firstname',
        'entity.email',
        search.createColumn({
            name: 'formulatext',
            formula: "{lastname}||', '||{firstname}"
        })
    ]
})

```

```

    ],

    // When the search is executed, the corresponding column in the result will then contain a value in the form: Last
    Name, First Name
    ...
    //Add additional code

```

Column.setWhenOrderedBy(options)

Method Description	<p>Returns the search column for which the minimal or maximal value should be found when returning the <code>search.Column</code> value.</p> <p>For example, can be set to find the most recent or earliest date, or the largest or smallest amount for a record, and then the <code>search.Column</code> value for that record is returned.</p> <div>  Note: You can only use this method if you use <code>MIN</code> or <code>MAX</code> as the summary type on a search column with the <code>Result.getValue(options)</code> method. </div>
Returns	<code>search.Column</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.name</code>	string	Required	The name of the search column for which the minimal or maximal value should be found.
<code>options.join</code>	string	Required	The join id for the search column.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
// Execute a customer search that returns the amount of the most recent sales order per customer

var filters = [];
var columns = [];
filters[0] = search.createFilter({
    name: 'recordtype',
    join: 'transaction',

```

```

        operator: search.Operator.IS,
        values: 'salesorder'
    });
    filters[1] = search.createFilter({
        name: 'mainline',
        join: 'transaction',
        operator: search.Operator.IS,
        values: true
    });
    columns[0] = search.createColumn({
        name: 'entityid',
        summary: search.Summary.GROUP
    });
    columns[1] = search.createColumn({
        name: 'totalamount',
        join: 'transaction',
        summary: search.Summary.MAX
    });
    columns[1].setWhenOrderedBy({
        name: 'trandate',
        join: 'transaction'
    });
    var mySearch = search.create({
        type: 'customer',
        filters: filters,
        columns: columns
    });
    var resultsArray = mySearch.run().getRange({
        start: 0,
        end: 100
    });
    ...
    //Add additional code

```

Column.name

Property Description	Name of a search column as a string.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
log.debug({

```



```

        details: 'Search Column Name: '
            + columnObj.name
    });
    ...
    //Add additional code

```

Column.join

Property Description	Join ID for a search column as a string.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
log.debug({
    details: 'Join ID for Search Column: '
        + columnObj.join
});
...
//Add additional code

```

Column.summary

Property Description	Returns the summary type for a search column.
Type	search.Summary enum
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...

```

```
log.debug({
  details: 'Summary Type for Search Column: '
    + columnObj.summary
});
...
//Add additional code
```

Column.formula

Property Description	Formula used for a search column as a string. To set this value, you must use <code>formulatext</code> , <code>formulanumeric</code> , <code>formuladatetime</code> , <code>formulapercent</code> , or <code>formulacurrency</code> .
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

For example, in the UI, a field with a custom UI label named **Customer Name** is set by a formula of type **Formula (Text)** and the formula is defined with the following formula:

```
//Add additional code
...
var columnObj = search.createColumn({
  name: 'formulatext',
  formula: "{firstname} || ', ' || {lastname}"
});
...
//Add additional code
```

In the above formula, `firstname` and `lastname` are script IDs for the fields on the Customer record form.

Column.label

Property Description	Label used for the search column. You can only get or set custom labels with this property.
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var columnObj = search.createColumn({
    name: 'formulanumeric',
    label: 'Numeric Formula'
});
...
//Add additional code
```

Column.function

Property Description	Special function applied to values in a search column. See Supported Functions .
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Supported Functions

The following table lists the supported functions and their internal IDs:

Internal ID	Name	Date Function	Output
percentOfTotal	% of Total	No	percent
absoluteValue	Absolute Value	No	
ageInDays	Age In Days	Yes	integer
ageInHours	Age In Hours	Yes	integer
ageInMonths	Age In Months	Yes	integer
ageInWeeks	Age In Weeks	Yes	integer
ageInYears	Age In Years	Yes	integer
calendarWeek	Calendar Week	Yes	date
day	Day	Yes	date
month	Month	Yes	text
negate	Negate	No	
numberAsTime	Number as Time	No	text
quarter	Quarter	Yes	text
rank	Rank	No	integer

Internal ID	Name	Date Function	Output
round	Round	No	
roundToHundredths	Round to Hundredths	No	
roundToTenths	Round to Tenths	No	
weekOfYear	Week of Year	Yes	text
year	Year	Yes	text

Errors

Error Code	Message	Thrown If
INVALID_SRCH_FUNCNTN	A search.Column contains an invalid function: {1}.	Unknown function is set.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var columnObj = search.createColumn({ // the age of the sales order in days
    name: 'trandate',
    function: 'ageInDays'
});
...
//Add additional code
```

Column.sort

Property Description	The sort order of the column. Use the search.Sort enum to set the value. If <code>Column.sort</code> is not set, the column is not sorted in any particular order.
Type	search.Sort enum
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var columnObj = search.createColumn({
```

```

    name: 'invoice',
    sort: search.Sort.DESC
  });
  ...
  //Add additional code

```

search.Filter

Object Description	<p>Encapsulates a search filter used in a search. Use the properties for the Filter object to get and set the filter properties.</p> <p>You create a search filter object with search.createFilter(options) and add it to a search.Search object that you create with search.create(options) or load with search.load(options).</p> <div>  Note: NetSuite uses an implicit AND operator with search filters, as opposed to filter expressions which explicitly use either AND and OR operators. </div> <p>Use the following guidelines with the Filter object:</p> <ul style="list-style-type: none"> ■ To search for a "none of null" value, meaning do not show results without a value for the specified field, use a value of @NONE@ in the Filter.formula property. ■ To search on checkbox fields, use the IS operator with a value of T or F to search for checked or unchecked fields, respectively. <p>For a complete list of this object's methods and properties, see Filter Object Members.</p>
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearchFilter = search.createFilter({
  name: 'entity',
  operator: search.Operator.ISEMPY,
});
...
//Add additional code

```

Filter.name

Property Description	Name or internal ID of the search field as a string. For more information, see search.createFilter(options) .
Type	string (read-only)
Supported Script Types	All script types

	For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: 'Filter Name: '
    + filterObj.name
});
...
//Add additional code
```

Filter.join

Property Description	Join ID for the search filter as a string.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: 'Join ID: '
    + filterObj.join
});
...
//Add additional code
```

Filter.operator

Property Description	Operator used for the search filter. This value is set with the search.Operator enum. The search.Operator enum contains the valid operator values for this property.
Type	string (read-only)

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearchFilter = search.createFilter({
    name: 'entity',
    operator: search.Operator.ISEMPY,
});
log.debug({
    details: 'Operator Used: '
        + mySearchFilter.operator
});
...
//Add additional code
```

Filter.summary

Property Description	Summary type for the search filter. Use this property to get or set the value of the summary type. See search.Summary .
Type	search.Summary
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_INVALID_SRCH_FILTER_SUM	A search.Filter contains an invalid summary type: {1}.	Unknown summary type is set.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
```

```

...
var mySearchFilter = search.createFilter({
    name: 'entity',
    summary: search.Summary.GROUP
});
...

//Add additional code

```

Filter.formula

Property Description	Formula used by the search filter. Use this property to get or set the formula used by the search filter. For more information about the formula property, see search.createFilter(options) .
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
log.debug({
    details: 'Search Filter Formula: '
        + filterObj.formula
});
...
//Add additional code

```

search.ResultSet

Object Description	Encapsulates a set of search results returned by Search.run() . Use the methods and properties for the ResultSet object to iterate through each result returned by the search or access an arbitrary slice of results, up to 1000 results at a time. For a complete list of this object's methods and properties, see ResultSet Object Members .
	Important: Search result sets are not cached. If records applicable to your search are created, modified, or deleted at the same time you are traversing your result set, your result set may change.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module

Since	2015.2
-------	--------

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
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;
});
...
//Add additional code
```

ResultSet.getRange(options)

Method Description	Retrieve a slice of the search result as an array of search.Result objects. The start parameter is the inclusive index of the first result to return. The end parameter is the exclusive index of the last result to return. For example, <code>getRange(0, 10)</code> retrieves 10 search results, at index 0 through index 9. Unlimited rows in the result are supported, however you can only return 1,000 at a time based on the index values. If there are fewer results available than requested, then the array will contain fewer than end - start entries. For example, if there are only 25 search results, then <code>getRange(20, 30)</code> will return an array of 5 search.Result objects.
Returns	search.Result[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/search Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.start</code>	number	Required	Index number of the first result to return, inclusive.

Parameter	Type	Required / Optional	Description
options.end	number	Required	Index number of the last result to return, exclusive.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var results = rs.getRange({
  start: 0,
  end: 1000
});
...
//Add additional code
```

ResultSet.getRange.promise(options)

Method Description	Method used to asynchronously retrieve a slice of the search result as an array of search.Result objects. <div> Note: For information about the parameters and errors thrown for this method, see ResultSet.getRange(options). For additional information on promises, see Promise Object.</div>
Returns	search.Result []
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
var results = rs.getRange.promise({
  start: 0,
  end: 1000
})
.then(function(response){
  log.debug({
    title: 'Completed',
    details: response
  });
});
```

```

    });
  })
  .catch(function onRejected(reason) {
    log.debug({
      title: 'Failed: ',
      details: reason
    });
  })
  ...

//Add additional code

```

ResultSet.each(callback)

Method Description	<p>Use a developer-defined function to invoke on each row in the search results, up to 4000 results at a time. The callback function must use the following signature:</p> <pre>boolean callback(result.Result result);</pre> <p>The callback function takes a search.Result object as an input parameter and returns a boolean which can be used to stop the iteration with a value of <code>false</code>, or continue the iteration with a value of <code>true</code>.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"> <p> Important: The work done in the context of the callback function counts towards the governance of the script that called it. For example, if the callback function is running in the context of a scheduled script, which has a 10,000 unit governance limit, make sure the amount of processing within the callback function does not put the entire script at risk of exceeding scheduled script governance limits.</p> </div>
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
callback	function	Required	Named JavaScript function or anonymous inline function that contains the logic to process a search.Result object.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
```

```

...
mySearch.run().each(function(result) {
    var entity = result.getValue({
        name: 'entity'
    });
    var subsidiary = result.getValue({
        name: 'subsidiary'
    });
    return true;
});
...
//Add additional code

```

ResultSet.each.promise(callback)

Method Description	Asynchronously uses a developer-defined function to invoke on each row in the search results, up to 4000 results at a time. The callback function must use the following signature: <div>  Note: For information about the parameters and errors thrown for this method, see ResultSet.each(callback). For additional information on promises, see Promise Object. </div>
Returns	void
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```

//Add additional code
...
mySearch.run().each.promise(function(result) {
    var entity = result.getValue({
        name: 'entity'
    });
    var subsidiary = result.getValue({
        name: 'subsidiary'
    });
    return true;
})
.then(function(response){
    log.debug({
        title: 'Completed',
        details: response
    });
})

```

```

        .catch(function onRejected(reason) {
            log.debug({
                title: 'Failed: ',
                details: reason
            });
        })
    ...
    //Add additional code

```

ResultSet.columns

Property Description	An array of search.Column objects that represent the columns returned in the search results.
Type	search.Column[] This property is read-only
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});

var resultSet = mySearch.run();
log.debug({
    details: "There are " + resultSet.columns.length + " columns in the result set:"
});

resultSet.columns.forEach(function(col){ // log each column
    log.debug({
        details: col
    });
});
...
//Add additional code

```

search.Page

Object Description	Encapsulates an individual search page containing a result set for a paginated search. For a complete list of this object's methods and properties, see Page Object Members .
---------------------------	---

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	Version 2015 Release 1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var page = pagedData.fetch({
    index: lastPageRange.index
});
...
//Add additional code
```

Page.next()

Method Description	Method used to fetch the next segment of data (bounded by search.PageRange). Moves the current page to next range.
Returns	Void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2016.1

Errors

Error Code	Message	Thrown If
INVALID_PAGE_RANGE	Invalid page range.	The page range is invalid, or when the page is the last page.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
while (!page.isFirst){
    page = page.next();
}
...
//Add additional code
```

Page.next.promise()

Method Description	<p>Method used to asynchronously fetch the next segment of data (bounded by search.PageRange).</p> <p>Moves the current page to another range. The promise is complete when the data for this range is loaded or rejected.</p> <div>  Note: For information about errors thrown for this method, see Page.next(). For additional information on promises, see Promise Object. </div>
Returns	Void
Synchronous Version	Page.next()
Supported Script Types	<p>All client-side scripts</p> <p>For more information, see SuiteScript 2.0 Client Script Type.</p>
Governance	5 units
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
return mypage.next.promise().then(processPage);
...
//Add additional code
```

Page.prev()

Method Description	<p>Method used to fetch the previous segment of data (bounded by search.PageRange).</p> <p>Moves the current page to previous range.</p>
Returns	Void
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Governance	5 units
Module	N/search Module
Since	2016.1

Errors

Error Code	Message	Thrown If
INVALID_PAGE_RANGE	Invalid page range.	The page range is invalid, or when the page is the first page.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
while (!page.isLast){
page = page.prev();
...
//Add additional code
```

Page.prev.promise()

Method Description	<p>Method used to asynchronously fetch the previous segment of data (bounded by search.PageRange).</p> <p>Moves the current page to another range. The promise is complete when the data for this range is loaded or rejected.</p> <div> Note: For information about errors thrown for this method, see Page.prev(). For additional information on promises, see Promise Object. </div>
Returns	Void
Synchronous Version	Page.prev()
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	5 units
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
return mypage.prev.promise().then(processPage);
...
//Add additional code
```

Page.data

Property Description	The results from a paginated search.
Type	search.Result[] This property is read-only.

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
function processPage(page){
    page.data.forEach(function(value){
        log.debug({
            details: "data: " + page.data
        });
    });
}
...
//Add additional code
```

Page.isFirst

Property Description	Indicates whether the page is within the first range of the result set. Flags the start of the data collection.
Type	boolean true false This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
while (!page.isFirst){
    page = page.next();
}
...
//Add additional code
```

Page.isLast

Property Description	Indicates whether a page is within the last range of the result set. Flags the end of the data collection.
-----------------------------	--

Type	boolean true false This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
while (!page.isLast){
  page = page.prev();
  ...
//Add additional code
```

Page.pagedData

Property Description	The <code>PagedData</code> Object used to fetch this <code>Page</code> Object.
Type	search.PagedData This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var lastPageRange = pagedData.pageRanges[pagedData.pageRanges.length - 1];
...
//Add additional code
```

Page.pageRange

Property Description	The <code>PageRange</code> Object used to fetch this <code>Page</code> Object. Page boundary information with the key and label.
Type	search.PageRange This property is read-only.

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: "Page Range: " + mySearchPage.pageRange
});
...

//Add additional code
```

search.PagedData

Object Description	<p>Holds metadata for a paginated query. This object provides a high-level view of a search result, giving the total count of records, a list of pages ranges, and page size. For a complete list of this object's methods and properties, see PagedData Object Members.</p> <div>  Important: Search result sets are not cached. If records applicable to your search are created, modified, or deleted at the same time you are traversing your result set, your result set may change. </div>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	Version 2015 Release 1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
// Run the paged search
var pagedData = mySearch.runPaged({
  pageSize: 1000
});
...
// Use the count property to count the
// search results easily
var resultCount = mySearch.runPaged({
```

```

    pageSize: 1000
  }).count;
  ...
  //Add additional code

```

PagedData.fetch(options)

Method Description	This method retrieves the data within the specified page range. This method also includes a promise version, <code>PagedData.fetch.promise()</code> . For more information about promises, see Promise Object .
Returns	search.Page
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>pageRange.index</code>	number	required	The index of the page range that bounds the desired data.

Errors

Error Code	Message	Thrown If
INVALID_PAGE_RANGE	Invalid page range.	The page range is not valid.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```

//Add additional code
...
var page = pagedData.fetch({
  index: lastPageRange.index
});
...
//Add additional code

```

PagedData.fetch.promise()

Method Description	This method asynchronously retrieves the data bounded by the <code>pageRange</code> parameter.
---------------------------	--

	 Note: For information about the parameters and errors thrown for this method, see PagedData.fetch(options) . For additional information on promises, see Promise Object .
Returns	search.Page
Synchronous Version	PagedData.fetch(options)
Supported Script Types	All client-side scripts For more information see, SuiteScript 2.0 Client Script Type .
Governance	5 units
Module	N/search Module
Since	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
return pagedData.fetch.promise().then(processPage);
...
//Add additional code
```

PagedData.count

Property Description	The total number of results when Search.runPaged(options) was executed.
Type	number This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: "Result Count: " + myPagedData.count
});
```

```
...
//Add additional code
```

PagedData.pageRanges

Property Description	The collection of PageRange objects that divide the entire result set into smaller groups. Includes page range information with the key and label for rendering.
Type	search.PageRange[] This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: "PageRange Array: " + myPagedData.pageRanges
});
...
//Add additional code
```

PagedData.pageSize

Property Description	Maximum number of entries per page Possible values are 5 - 1000 entries per page.
Type	number This is a read-only property.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
```

```
log.debug({
    details: "Max Page Size: " + myPagedData.pageSize
});
...
//Add additional code
```

PagedData.searchDefinition

Property Description	The search criteria used to execute the result set for this <code>PagedData</code> Object.
Type	read-only search.Search
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
    details: "Search Details: " + myPagedData.searchDefinition
});
...
//Add additional code
```

search.PageRange

Object Description	Defines the page range to contain the result set For a complete list of this object's properties, see PageRange Object Members .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	Version 2015 Release 1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var page = pagedData.fetch({
    index: lastPageRange
});
```

```
...
//Add additional code
```

PageRange.compoundLabel

Property Description	Human-readable label with beginning and ending range identifiers
Type	read-only string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: "Page Range Description: " + myPageRange.compoundLabel
});
...
//Add additional code
```

PageRange.index

Property Description	The index of the pageRange
Type	number This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
log.debug({
  details: "Page Range Index: " + myPageRange.index
});
...
```



```
//Add additional code
```

search.Setting

Object Description	<p>Defines a search setting.</p> <p>Search settings let you specify search parameters that are typically available only in the UI. The following settings are supported:</p> <ul style="list-style-type: none"> ■ Consolidated Exchange Rate: This setting affects how consolidation is performed (for example, consolidation using the Average rate type, consolidation using the Historical rate type, and so on). This setting applies to transaction searches, and it is applicable only to OneWorld accounts. ■ Show Period End Transactions: This setting indicates whether period end transactions are included in search results. This setting applies to transaction searches. <p>Use search.createSetting(options) to create a setting. After you create your settings, assign them as array values to Search.settings.</p> <p>For a complete list of this object's properties, see Setting Object Members.</p>
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/search Module
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
  type: 'transaction',
  columns: [ 'trandate', 'amount', 'entity' ],
  filters: [
    search.createFilter({
      name: 'internalid',
      operator: search.Operator.ANYOF,
      values: [13, 12356]
    })
  ],
  settings: [
    search.createSetting({
      name: 'consolidationtype',
      value: 'NONE'
    })
  ]
});
...
//Add additional code
```

Setting.name

Property Description	The name of the search parameter.
-----------------------------	-----------------------------------

	<p>This property is set when you call search.createSetting(options). The following values are supported for this property:</p> <ul style="list-style-type: none"> ■ <code>consolidationtype</code>: This value corresponds to the Consolidated Exchange Rate setting. ■ <code>includeperiodendtransactions</code>: This value corresponds to the Show Period End Transactions setting.
Type	<p>string</p> <p>This property is read-only.</p>
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/search Module
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
  type: 'transaction',
  columns: [ 'trandate', 'amount', 'entity' ],
  filters: [
    search.createFilter({
      name: 'internalid',
      operator: search.Operator.ANYOF,
      values: [13, 12356]
    })
  ],
  settings: [
    search.createSetting({
      name: 'consolidationtype',
      value: 'NONE'
    })
  ]
});
...
//Add additional code
```

Setting.value

Property Description	<p>The value of the search parameter.</p> <p>This property is set when you call search.createSetting(options). If you specify <code>consolidationtype</code> as the search parameter name (Setting.name), the following values are supported for this parameter:</p> <ul style="list-style-type: none"> ■ ACCTTYPE ■ AVERAGE ■ CURRENT ■ HISTORICAL ■ NONE <p>If you specify <code>includeperiodendtransactions</code> as the search parameter name (Setting.name), the following values are supported for this parameter:</p>
----------------------	---

	<ul style="list-style-type: none"> ■ F ■ FALSE ■ T ■ TRUE <p>These values are not case sensitive.</p>
Type	string This property is read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2018.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
  type: 'transaction',
  columns: [ 'trandate', 'amount', 'entity' ],
  filters: [
    search.createFilter({
      name: 'internalid',
      operator: search.Operator.ANYOF,
      values: [13, 12356]
    })
  ],
  settings: [
    search.createSetting({
      name: 'consolidationtype',
      value: 'NONE'
    })
  ]
});
...
//Add additional code
```

search.create(options)

Method Description	<p>Creates a new search and returns it as a search.Search object.</p> <p>The search can be modified and run as an on demand search with Search.run(), without saving it. Alternatively, calling Search.save() will save the search to the database, so it can be reused later in the UI or loaded with search.load(options).</p> <div> Note: This method is agnostic in terms of its <code>options.filters</code> argument. It can accept input of a single search.Filter object, an array of search.Filter objects, or a search filter expression. </div> <p>The <code>search.create(options)</code> method also includes a promise version, <code>search.create.promise(options)</code>. For more information about promises, see Promise Object.</p>
--------------------	--

	 Important: When you use this method to create a search, consider the following: <ul style="list-style-type: none"> When you define the search, make sure you sort using the field with the most unique values, or sort using multiple fields. Sorting with a single field that has multiple identical values can cause the result rows to be in a different order each time the search is run. You cannot directly create a filter or column for a list/record type field in SuiteScript by passing in its text value. You must use the field's internal ID. If you must use the field's text value, you can create a filter or column with a formula using <code>name : 'formulatext'</code>.
Returns	search.Search
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	string	Required	The search type that you want to base the search on. Use the search.Type enum for this argument.	2015.2
<code>options.filters</code>	search.Filter[] Object[]	Optional	<p>A single search.Filter object, an array of search.Filter objects, a search filter expression, or an array of search filter expressions.</p> <p>A search filter expression can be passed in as an Object with the following properties:</p> <ul style="list-style-type: none"> name (required) join operator (required) summary formula <p>For more information about these properties, see Filter Object Members.</p> <div>  Note: You can further filter the returned search.Search object by adding additional filters with Search.filters or Search.filterExpression. </div>	2015.2
<code>options.filterExpression</code>	Object[]	Optional	Search filter expression for the search as an array of expression objects. A search filter expression is a JavaScript string array of zero or more elements. Each element is one of the following:	2016.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> Operator - either 'NOT', 'AND', or 'OR' Filter term Nested search filter expression <p>You set this value with an array of expression objects or single filter expression object to overwrite any prior filter expressions. Use <code>null</code> to set an empty array and remove any existing filter expressions on this search.</p> <div>  Note: If you want to get or set a search filters, use the <code>Search.filters</code> property. </div> <p>This parameter sets the value for the <code>Search.filterExpression</code> property.</p>	
<code>options.columns</code>	<code>search.Column[]</code> <code>Object[]</code>	Optional	<p>A single <code>search.Column</code> object or array of <code>search.Column</code> objects. You can optionally pass in an Object or array of Objects with the following properties to represent a Column:</p> <ul style="list-style-type: none"> name (required) formula function join label sort summary <p>For more information about these properties, see Column Object Members.</p>	2015.2
<code>options.settings</code>	<code>search.Setting[]</code>	Optional	Search settings for this search as an array of <code>search.Setting</code> objects. Search settings let you specify search parameters that are typically available only in the UI. See Search.settings .	2018.2
<code>options.title</code>	string	Optional	The name for a saved search. The title property is required to save a search with <code>Search.save()</code> .	2015.2
<code>options.id</code>	string	Optional	Script ID for a saved search. If you do not set the saved search ID, NetSuite generates one for you. See Search.id .	2015.2
<code>options.isPublic</code>	boolean <code>true</code> <code>false</code>	Optional	Set to <code>true</code> to make the search public. Otherwise, set to <code>false</code> . If you do not set this parameter, it defaults to <code>false</code> . This parameter sets the value for the <code>Search.isPublic</code> property.	2016.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.

Error Code	Message	Thrown If
SSS_INVALID_SRCH_FILTER_EXPR	Malformed search filter expression. This is a general error raised when a filter expression cannot be parsed. For example: [f1, 'and', 'and', f2]	The <code>options.filters</code> parameter is not a valid search filter, filter array, or filter expression.
SSS_INVALID_SRCH_COL	An <code>search.Column</code> Object contains an invalid column, or is not in proper syntax: {1}.	The <code>options.columns</code> parameter is not a valid column, string, or column or string array.
SSS_INVALID_SRCH_SETTING		An unknown search parameter name is provided.
SSS_INVALID_SRCH_SETTING_VALUE		An unsupported value is set for the provided search parameter name.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySalesOrderSearch = search.create({
  type: search.Type.SALES_ORDER,
  title: 'My Second SalesOrder Search',
  id: 'customsearch_my_second_so_search',
  columns: [{
    name: 'entity'
  }, {
    name: 'subsidiary'
  }, {
    name: 'name'
  }, {
    name: 'currency'
  }],
  filters: [{
    name: 'mainline',
    operator: 'is',
    values: ['T']
  }],
  settings: [{
    name: 'consolidationtype',
    value: 'AVERAGE'
  }]
});
...
//Add additional code
```

search.create.promise(options)

Method Description	Creates a new search asynchronously and returns it as a search.Search object. <div>  Note: For information about the parameters and errors thrown for this method, see search.create(options). For additional information on promises, see Promise Object. </div>
Returns	search.Search
Synchronous Version	search.create(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.create.promise({
    type: search.Type.SALES_ORDER
})
.then(function(result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function(reason) {
    log.debug({
        details: "Failed: " + reason
    })
    // do something on failure
});
...
//Add additional code
```

search.createSetting(options)

Method Description	Creates a new search setting and returns it as a search.Setting object. Search settings let you specify search parameters that are typically available only in the UI. The following settings are supported:
---------------------------	---

	<ul style="list-style-type: none"> ■ Consolidated Exchange Rate: This setting affects how consolidation is performed (for example, consolidation using the Average rate type, consolidation using the Historical rate type, and so on). This setting applies to transaction searches, and it is applicable only to OneWorld accounts. ■ Show Period End Transactions: This setting indicates whether period end transactions are included in search results. This setting applies to transaction searches. <p>After you create your settings, assign them as array values to Search.settings.</p>
Returns	search.Setting
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	
Module	N/search Module
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.name</code>	string	Required	<p>The name of the search parameter to set. This value sets the Setting.name property.</p> <p>Use one of the following values for this parameter:</p> <ul style="list-style-type: none"> ■ <code>consolidationtype</code>: This value corresponds to the Consolidated Exchange Rate setting. ■ <code>includeperiodendtransactions</code>: This value corresponds to the Show Period End Transactions setting. 	2018.2
<code>options.value</code>	string	Required	<p>The value of the search parameter. If you are executing a joined search, this value is the join ID used for the search field specified by the <code>options.name</code> parameter. This value sets the Setting.value property.</p> <p>If you specify <code>consolidationtype</code> as the search parameter name, use one of the following values for this parameter:</p> <ul style="list-style-type: none"> ■ <code>ACCTTYPE</code> ■ <code>AVERAGE</code> ■ <code>CURRENT</code> ■ <code>HISTORICAL</code> ■ <code>NONE</code> <p>The default value is <code>ACCTTYPE</code>, which represents the type of consolidation associated with the account.</p> <p>If you specify <code>includeperiodendtransactions</code> as the search parameter name, use one of the following values for this parameter:</p> <ul style="list-style-type: none"> ■ <code>F</code> ■ <code>FALSE</code> ■ <code>T</code> ■ <code>TRUE</code> <p>The default value is <code>false</code>.</p> <p>These values are not case sensitive.</p>	2018.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	A required parameter is missing.
SSS_INVALID_SRCH_SETTING		An unknown search parameter name is provided.
SSS_INVALID_SRCH_SETTING_VALUE		An unsupported value is set for the provided search parameter name.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
  type: 'transaction',
  columns: [ 'trandate', 'amount', 'entity' ],
  filters: [
    search.createFilter({
      name: 'internalid',
      operator: search.Operator.ANYOF,
      values: [13, 12356]
    })
  ],
  settings: [
    search.createSetting({
      name: 'consolidationtype',
      value: 'NONE'
    })
  ]
});
...
//Add additional code
```

search.load(options)

Method Description	Loads an existing saved search and returns it as a search.Search . The saved search could have been created using the UI or created with search.create(options) and Search.save() . The search.load(options) method also includes a promise version, search.load.promise(options) . For more information about promises, see Promise Object .
Returns	search.Search
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	Required	Internal ID or script ID of a saved search. The script ID starts with <code>customsearch</code> . See Search.id .	2015.2
options.type	string	Required if the saved search to load uses a standalone search type, optional otherwise	<p>The search type of the saved search to load. Use a value from the search.Type enum for this parameter. This parameter is required if the saved search to load uses a standalone search type. A standalone search type is a search type that does not have a corresponding record type. Typically, the search type of the saved search can be determined automatically based on the corresponding record type. In this case, this parameter is not required. For standalone search types, you must specify the search type explicitly using this parameter. The following is a list of standalone search types:</p> <ul style="list-style-type: none"> ■ DeletedRecord ■ EndToEndTime ■ ExpenseAmortPlanAndSchedule ■ RevRecPlanAndSchedule ■ GLLinesAuditLog ■ Crosschargeable ■ FinRptAggregateFR ■ BillingAccountBillCycle ■ BillingAccountBillRequest ■ BinItemBalance ■ PaymentEvent ■ Permission ■ GatewayNotification ■ TimeApproval ■ RecentRecord ■ Role ■ SavedSearch ■ ShoppingCart ■ SubscriptionRenewalHistory ■ SuiteScriptDetail ■ SupplyChainSnapshotDetails ■ SystemNote ■ TaxDetail ■ TimesheetApproval ■ Uber ■ ResAllocationTimeOffConflict ■ ComSearchOneWaySyn ■ ComSearchGroupSyn ■ Installment ■ InventoryBalance ■ InventoryNumberBin 	2015.2

Parameter	Type	Required / Optional	Description	Since
			<ul style="list-style-type: none"> InventoryNumberItem InventoryStatusLocation InvntNumberItemBalance ItemBinNumber 	

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.
INVALID_SEARCH	That search or mass update does not exist.	Cannot find saved search with the saved search ID from <code>options.id</code> parameter.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.load({
    id: 'customsearch_my_so_search'
});
...
//Add additional code
```

search.load.promise(options)

Method Description	<p>Loads an existing saved search asynchronously and returns it as a search.Search object. The saved search could have been created using the UI or created with search.create(options) and Search.save().</p> <div>  Note: For information about the parameters and errors thrown for this method, see search.load(options). For additional information on promises, see Promise Object. </div>
Returns	search.Search
Synchronous Version	search.load(options)
Supported Script Types	<p>All client-side scripts</p> <p>For more information, see SuiteScript 2.0 Client Script Type.</p>
Governance	5 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.load.promise({
    type : search.Type.SALES_ORDER,
    id : 'customsearch_txn_search_salesorder'
})
.then(function (result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function onRejected(reason) {
    // do something on rejection
});
...
//Add additional code
```

search.delete(options)

Method Description	Deletes an existing saved search. The saved search could have been created using the UI or created with search.create(options) and Search.save() . The search.delete(options) method also includes a promise version, search.delete.promise(options) . For more information about promises, see Promise Object .
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	5 units
Module	N/search Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	Required	Internal ID or script ID of a saved search. The script ID starts with <code>customsearch</code> . See Search.id .	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.

Error Code	Message	Thrown If
INVALID_SEARCH	That search or mass update does not exist.	Cannot find saved search with the saved search ID from <code>options.id</code> parameter.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
search.delete({
  id: 'customsearch_my_so_search'
});
...
//Add additional code
```

search.delete.promise(options)

Method Description	Deletes an existing saved search asynchronously and returns it as a <code>search.Search</code> object. The saved search can be created using the UI or created with <code>search.create(options)</code> and <code>Search.save()</code> . <div data-bbox="440 989 1370 1104" style="border: 1px solid #0070c0; padding: 5px; margin-top: 10px;">  Note: For information about the parameters and errors thrown for this method, see <code>search.delete(options)</code>. For additional information on promises, see Promise Object. </div>
Returns	void
Synchronous Version	<code>search.delete(options)</code>
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	5 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.delete.promise({id: 'customsearch_txn_search_salesorder'})
  .then(function(){
    search.load({
      id: 'customsearch_txn_search_salesorder'
    });
  });
})
```

```

.catch(function onRejected(reason) {
    log.debug({
        details: 'Invalid search: ' reason.name
    });
});
...
//Add additional code

```

search.duplicates(options)

Method Description	<p>Performs a search for duplicate records based on the account's duplicate detection configuration.</p> <p>This method also includes a promise version, search.duplicates.promise(options). For more information about promises, see Promise Object.</p> <div style="border: 1px solid #ffc107; padding: 5px; margin: 10px 0;">  Important: This API works only for records that support duplicate record detection (for example, customer, lead, prospect, contact, partner, and vendor records). </div> <p>For more information about duplicate record detection, see the help topic Duplicate Record Detection.</p>
Returns	<p>search.Result[] that contains the duplicate records</p> <p>Results are limited to 1000 rows.</p> <p>If there are no search results, this method returns <code>null</code>.</p>
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Governance	10 units
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.type</code>	enum	Required	The search type that you want to check for duplicates. Use the search.Type enum for this parameter. The type you specify must correspond to a record type that supports duplicate record detection (for example, customer, lead, prospect, contact, partner, and vendor records).	2015.2
<code>options.fields</code>	Object	Optional	<p>A set of key/value pairs used to detect duplicates. The keys are internal ID names of the fields used to detect duplicates. You can specify fields such as companyname, email, name, phone, address1, city, state, and zipcode. For example, to detect duplicates based on the value of the email field, use 'email' : 'sample@test.com'.</p> <p>Use this parameter to specify the fields (and their values) to use to detect duplicates. If you are searching for duplicates based on fields that appear on a certain record type, this parameter is required. If you are searching for duplicates of a specific record (of the specified type), use the <code>options.id</code> parameter instead.</p>	2015.2
<code>options.id</code>	number	Optional	Internal ID of an existing record.	2015.2

Parameter	Type	Required / Optional	Description	Since
			Use this parameter to specify a record to detect duplicates of. The duplicate record detection settings in the account determine which fields are used to detect duplicates of the specified record. If you are searching for duplicates of a specific record (of the specified type), this parameter is required. If you are searching for duplicates based on fields that appear on a certain record type, use the options.fields parameter instead.	

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
// Search for duplicates of a specific record using the options.id
// parameter
var duplicatesOfRecord = search.duplicates({
    type: search.Type.CONTACT,
    id: 425
});

// Search for duplicates based on specific fields on a record type
// using the options.fields parameter
var duplicatesUsingFields = search.duplicates({
    type: search.Type.CONTACT,
    fields: {
        'email' : 'sample@test.com'
    }
});
...
//Add additional code
```

search.duplicates.promise(options)

Method Description	<p>Performs a search for duplicate records asynchronously based on the Duplicate Detection configuration for the account. Returns an array of search.Result objects. This method only applies to records that support duplicate record detection. These records include customer lead prospect partner vendor contact.</p> <div> Note: For information about the parameters and errors thrown for this method, see search.duplicates(options). For additional information on promises, see Promise Object. </div>
Returns	search.Result []

Synchronous Version	search.duplicates(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.duplicates.promise({
    type: search.Type.CUSTOMER,
    id: 28
})
.then(function (result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function onRejected(reason) {
    // do something on rejection
});
...
//Add additional code
```

search.global(options)

Method Description	Performs a global search against a single keyword or multiple keywords. Similar to the global search functionality in the UI, you can programmatically filter the global search results that are returned. For example, you can use the following filter to limit the returned records to Customer records: <code>'cu: simpson'</code> The <code>search.global(options)</code> method also includes a promise version, <code>search.global.promise(options)</code> . For more information about promises, see Promise Object . For more information about global search, see the help topic Global Search .
Returns	search.Result[] as an array of result objects containing these columns: name, type, info1, and info2 Results are limited to 1000 records. If there are no search results, this method returns <code>null</code> .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/search Module

Since	2015.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.keywords</code>	string	Required	Global search keywords string or expression.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var customerSearch = search.global({
    keywords: 'cu: simpson'
});
...
//Add additional code
```

search.global.promise(options)

Method Description	<p>Performs a global search asynchronously against a single keyword or multiple keywords. Returns an array of search.Result objects with four columns: name, type, info1, and info2.</p> <div>  Note: For information about the parameters and errors thrown for this method, see search.global(options). For additional information on promises, see Promise Object. </div>
Returns	search.Result []
Synchronous Version	search.global(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/search Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.global.promise({
    keywords: 'Alan Rath'
})
.then(function (result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function onRejected(reason) {
    // do something on rejection
});
...
//Add additional code
```

search.lookupFields(options)

Method Description	<p>Performs a search for one or more body fields on a record. You can use joined-field lookups with this method, with the following syntax:</p> <pre>join_id.field_name</pre> <p>The <code>search.lookupFields(options)</code> method also includes a promise version, <code>search.lookupFields.promise(options)</code>. For more information about promises, see Promise Object.</p> <p>Note that the return contains either an object or a scalar value, depending on whether the looked-up field holds a single value, or a collection of values. Single select fields are returned as an object with value and text properties. Multi-select fields are returned as an object with value: text pairs.</p> <p>In the following example, a select field like <code>my_select</code> would return an array of objects containing a value and text property. This select field contains multiple entries to select from, so each entry would have a numerical id (the value) and a text display (the text). For "internalid" in this particular code snippet, the sample returns 1234. The internal id of a record is a single value, so a scalar is returned.</p>
Returns	<p>Object array</p> <ul style="list-style-type: none"> ■ Returns select fields as an object with value and text properties.

```
{
  internalid: 1234,
  firstname: 'Joe',
  my_select: [{
    value: 1,
    text: 'US Sub'
  }],
  my_multiselect: [{
    "value": "1,2",
    "text": "US Sub, EU Sub"
  }]
}
```

	<ul style="list-style-type: none"> Returns multiselect fields as an array of object with value:text pairs.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	1 unit
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	enum	Required	The search type for which you want to look up fields. Use the search.Type enum for this argument.	2015.2
options.id	string	Required	Internal ID for the record, for example 777 or 87.	2015.2
options.columns	string string[]	Required	Array of column/field names to look up, or a single column/field name. The <code>columns</code> parameter can also be set to reference joined fields.	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var fieldLookUp = search.lookupFields({
  type: search.Type.SALES_ORDER,
  id: '87',
  columns: ['entity', 'subsidiary', 'name', 'currency']
});
...
//Add additional code
```

search.lookupFields.promise(options)

Method Description	Performs a search asynchronously for one or more body fields on a record. Returns select fields as an object with value and text properties. Returns multiselect fields as an object with value:text pairs.
---------------------------	---

	 Note: For information about the parameters and errors thrown for this method, see search.lookupFields(options) . For additional information on promises, see Promise Object .
Returns	object array
Synchronous Version	search.lookupFields(options)
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	1 unit
Module	N/search Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
search.lookupFields.promise({
    type: search.Type.EMPLOYEE,
    id: -5,
    columns : 'email'
})
.then(function (result) {
    log.debug({
        details: "Completed: " + result
    });
    // do something after completion
})
.catch(function onRejected(reason) {
    // do something on rejection
});
...
//Add additional code
```

search.createColumn(options)

Method Description	Creates a new search column as a search.Column object. <div>  Important: You cannot directly create a filter or column for a list/record type field in SuiteScript by passing in its text value. You must use the field's internal ID. If you must use the field's text value, you can create a filter or column with a formula using <code>name: 'formulatext'</code>. </div>
Returns	search.Column
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.name	string	Required	Name of the search column. See Column.name .	2015.2
options.join	string	Optional	Join ID for the search column. See Column.join .	2015.2
options.summary	enum	Optional	Summary type for the column. See search.Summary and Column.summary .	2015.2
options.formula	string	Optional	Formula for the search column. See Column.formula .	2015.2
options.function	string	Optional	Special function for the search column. See Column.function .	2015.2
options.label	string	Optional	Label for the search column. See Column.label .	2015.2
options.sort	enum	Optional	The sort order of the column. Use the search.Sort enum for this argument. Also see Column.sort .	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.
SSS_INVALID_SRCH_COLUMN_SUM	A search.Column object contains an invalid column summary type, or is not in proper syntax: {1}.	The options.summary parameter is not a valid search summary type. See search.Summary .
INVALID_SRCH_FUNCNTN		An unknown function is provided.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var currencyColumn = search.createColumn({
  name: 'currency',
  sort: search.Sort.ASC
});
```

```
...
//Add additional code
```

search.createFilter(options)

Method Description	Creates a new search filter as a search.Filter object. <div> Important: You cannot directly create a filter or column for a list/record type field in SuiteScript by passing in its text value. You must use the field's internal ID. If you must use the field's text value, you can create a filter or column with a formula using <code>name: 'formulatext'</code>.</div>
Returns	search.Filter
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/search Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.name</code>	string	Required	Name or internal ID of the search field.	2015.2
<code>options.join</code>	string	Optional	Join ID for the search filter.	2015.2
<code>options.operator</code>	search.Operator	Required	Operator used for the search filter. Use the search.Operator enum.	2015.2
<code>options.values</code>	string Date number string[] Date[]	Optional	Values to be used as filter parameters.	2015.2
<code>options.formula</code>	string	Optional	Formula used by the search filter.	2015.2
<code>options.summary</code>	search.Summary	Optional	Summary type for the search filter. See search.Summary .	2015.2

Errors

Error Code	Message	Thrown If
SSS_MISSING_REQD_ARGUMENT	{1}: Missing a required argument: {2}	Required parameter is missing.
SSS_INVALID_SRCH_FILTER_SUM	A search.Column object contains an invalid column summary type, or is not in proper syntax: {1}.	<code>options.summary</code> parameter is not a valid search summary type. See search.Summary .

Error Code	Message	Thrown If
SSS_INVALID_SRCH_OPERATOR	An search.Filter object contains an invalid operator, or is not in proper syntax: {1}.	<code>options.operator</code> parameter is not a valid operator type. See search.Operator .

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearchFilter = search.createFilter({
    name: 'entity',
    operator: search.Operator.ISEMPY
});
...
//Add additional code
```

search.Operator

Enum Description	<p>Enumeration that holds the values for search operators to use with the search.Filter. See the help topic Search Operators for more information about the field types supported for each operator type.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Values

<ul style="list-style-type: none"> ■ AFTER ■ ALLOF ■ ANY ■ ANYOF ■ BEFORE ■ BETWEEN ■ CONTAINS ■ DOESNOTCONTAIN ■ DOESNOTSTARTWITH ■ EQUALTO ■ GREATERTHAN 	<ul style="list-style-type: none"> ■ IS ■ ISEMPY ■ ISNOT ■ ISNOTEMPTY ■ LESSTHAN ■ LESSTHANOREQUALTO ■ NONEOF ■ NOTAFTER ■ NOTALLOF ■ NOTBEFORE ■ NOTBETWEEN 	<ul style="list-style-type: none"> ■ NOTGREATERTHANOREQUALTO ■ NOTLESSTHAN ■ NOTLESSTHANOREQUALTO ■ NOTON ■ NOTONORAFTER ■ NOTONORBEFORE ■ NOTWITHIN ■ ON ■ ONORAFTER ■ ONORBEFORE ■ STARTSWITH
---	---	--

<ul style="list-style-type: none"> GREATERTHANOREQUALTO HASKEYWORDS 	<ul style="list-style-type: none"> NOTEQUALTO NOTGREATERTHAN 	<ul style="list-style-type: none"> WITHIN
---	--	--

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code...
var mySearchFilter = search.createFilter({
    name: 'entity',
    operator: search.Operator.ISEMPY
});
...

//Add additional code
```

search.Sort

Enum Description	Enumeration that holds the values for supported sorting directions used with <code>search.createColumn(options)</code> . <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Values

- ASC
- DESC
- NONE

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var currencyColumn = search.createColumn({
    name: 'currency',
    sort: search.Sort.ASC
```



```
});
...
//Add additional code
```

search.Summary

Enum Description	Enumeration that holds the values for summary types used by the Column.summary or Filter.summary properties. For more information about each summary type, see the help topic Search Summary Types . <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Values

- GROUP
- COUNT
- SUM
- AVG
- MIN
- MAX

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearchFilter = search.createFilter({
  name: 'entity',
  summary: search.Summary.GROUP
});
...
//Add additional code
```

search.Type

Enum Description	Enumeration that holds the string values for search types supported in the N/search Module . This enum is used to pass the type argument to search.create(options) .
-------------------------	--

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/search Module
Since	2015.2

Values

 **Note:** A search type is not a synonym for a record type. The supported search types listed below do not necessarily correspond with the supported record types listed in the [N/record Module](#).

<ul style="list-style-type: none"> ■ ACCOUNT ■ ACCOUNTING_BOOK ■ ACCOUNTING_CONTEXT ■ ACCOUNTING_PERIOD ■ ACTIVITY ■ ADV_INTER_COMPANY_JOURNAL_ENTRY ■ AMORTIZATION_SCHEDULE ■ AMORTIZATION_TEMPLATE ■ ASSEMBLY_BUILD ■ ASSEMBLY_ITEM ■ ASSEMBLY_UNBUILD ■ BILLING_ACCOUNT ■ BILLING_ACCOUNT_BILL_CYCLE ■ BILLING_ACCOUNT_BILL_REQUEST ■ BILLING_CLASS ■ BILLING_RATE_CARD ■ BILLING_REVENUE_EVENT ■ BILLING_SCHEDULE ■ BIN ■ BIN_TRANSFER ■ BIN_WORKSHEET ■ BLANKET_PURCHASE_ORDER ■ BOM ■ BOM_REVISION ■ BUNDLE_INSTALLATION_SCRIPT ■ CALENDAR_EVENT ■ CAMPAIGN ■ CASH_REFUND ■ CASH_SALE 	<ul style="list-style-type: none"> ■ FIN_RPT_AGGREGATE_F_R ■ FIXED_AMOUNT_PROJECT_REVENUE_RULE ■ FOLDER ■ FULFILLMENT_REQUEST ■ GENERIC_RESOURCE ■ GIFT_CERTIFICATE ■ GIFT_CERTIFICATE_ITEM ■ GLOBAL_ACCOUNT_MAPPING ■ GLOBAL_INVENTORY_RELATIONSHIP ■ GL_LINES_AUDIT_LOG ■ INBOUND_SHIPMENT ■ INTER_COMPANY_JOURNAL_ENTRY ■ INTER_COMPANY_TRANSFER_ORDER ■ INVENTORY_ADJUSTMENT ■ INVENTORY_BALANCE ■ INVENTORY_COST_REVALUATION ■ INVENTORY_COUNT ■ INVENTORY_DETAIL ■ INVENTORY_ITEM ■ INVENTORY_NUMBER ■ INVENTORY_STATUS ■ INVENTORY_STATUS_CHANGE ■ INVENTORY_TRANSFER ■ INVOICE ■ ISSUE ■ ITEM ■ ITEM_ACCOUNT_MAPPING 	<ul style="list-style-type: none"> ■ PROJECT_TEMPLATE ■ PROMOTION_CODE ■ PROSPECT ■ PURCHASE_CONTRACT ■ PURCHASE_ORDER ■ PURCHASE_REQUISITION ■ RECENT_RECORD ■ RES_ALLOCATION_TIME_OFF_CONFLICT ■ RESOURCE_ALLOCATION ■ RESTLET ■ RETURN_AUTHORIZATION ■ REVENUE_ARRANGEMENT ■ REVENUE_COMMITMENT ■ REVENUE_COMMITMENT_REVERSAL ■ REVENUE_PLAN ■ REV_REC_SCHEDULE ■ REV_REC_TEMPLATE ■ ROLE ■ SALES_ORDER ■ SALES_ROLE ■ SALES_TAX_ITEM ■ SAVED_SEARCH ■ SCHEDULED_SCRIPT ■ SCHEDULED_SCRIPT_INSTANCE ■ SCRIPT_DEPLOYMENT ■ SERIALIZED_ASSEMBLY_ITEM ■ SERIALIZED_INVENTORY_ITEM ■ SERVICE_ITEM ■ SHIP_ITEM
--	---	---

■ CHARGE	■ ITEM_BIN_NUMBER	■ SOLUTION
■ CHARGE_RULE	■ ITEM_DEMAND_PLAN	■ STATISTICAL_JOURNAL_ENTRY
■ CHECK	■ ITEM_FULFILLMENT	■ STORE_PICKUP_FULFILLMENT
■ CLASSIFICATION	■ ITEM_GROUP	■ SUBSCRIPTION
■ CLIENT_SCRIPT	■ ITEM_RECEIPT	■ SUBSCRIPTION_CHANGE_ORDER
■ CMS_CONTENT	■ ITEM_REVISION	■ SUBSCRIPTION_LINE
■ CMS_CONTENT_TYPE	■ ITEM_SUPPLY_PLAN	■ SUBSCRIPTION_PLAN
■ COM_SEARCH_GROUP_SYN	■ JOB	■ SUBSCRIPTION_RENEWAL_HISTORY
■ COM_SEARCH_ONE_WAY_SYN	■ JOB_STATUS	■ SUBSIDIARY
■ COMMERCE_CATEGORY	■ JOB_TYPE	■ SUBTOTAL_ITEM
■ COMPETITOR	■ JOURNAL_ENTRY	■ SUITELET
■ CONSOLIDATED_EXCHANGE_RATE	■ KIT_ITEM	■ SUITE_SCRIPT_DETAIL
■ CONTACT	■ LABOR_BASED_PROJECT_REVENUE_RULE	■ SUPPLY_CHAIN_SNAPSHOT
■ CONTACT_CATEGORY	■ LEAD	■ SUPPORT_CASE
■ CONTACT_ROLE	■ LOCATION	■ SYSTEM_NOTE
■ COST_CATEGORY	■ LOT_NUMBERED_ASSEMBLY_ITEM	■ TASK
■ COUPON_CODE	■ LOT_NUMBERED_INVENTORY_ITEM	■ TAX_DETAIL
■ CREDIT_CARD_CHARGE	■ MANUFACTURING_COST_TEMPLATE	■ TAX_GROUP
■ CREDIT_CARD_REFUND	■ MANUFACTURING_OPERATION_TASK	■ TAX_PERIOD
■ CREDIT_MEMO	■ MANUFACTURING_ROUTING	■ TAX_TYPE
■ CURRENCY	■ MAP_REDUCE_SCRIPT	■ TERM
■ CUSTOMER	■ MARKUP_ITEM	■ TIME_APPROVAL
■ CUSTOMER_CATEGORY	■ MASSUPDATE_SCRIPT	■ TIME_BILL
■ CUSTOMER_DEPOSIT	■ MERCHANDISE_HIERARCHY_LEVEL	■ TIME_OFF_CHANGE
■ CUSTOMER_MESSAGE	■ MERCHANDISE_HIERARCHY_NODE	■ TIME_OFF_PLAN
■ CUSTOMER_PAYMENT	■ MERCHANDISE_HIERARCHY_VERSION	■ TIME_OFF_REQUEST
■ CUSTOMER_PAYMENT_AUTHORIZATION	■ MESSAGE	■ TIME_OFF_RULE
■ CUSTOMER_REFUND	■ MFG_PLANNED_TIME	■ TIME_OFF_TYPE
■ CUSTOMER_STATUS	■ NEXUS	■ TOPIC
■ CUSTOM_RECORD	■ NON_INVENTORY_ITEM	■ TRANSACTION
■ CUSTOM_TRANSACTION	■ NOTE	■ TRANSFER_ORDER
■ DELETED_RECORD	■ NOTE_TYPE	■ UBER
■ DEPARTMENT	■ OPPORTUNITY	■ UNITS_TYPE
■ DEPOSIT	■ OTHER_CHARGE_ITEM	■ USAGE
■ DEPOSIT_APPLICATION	■ OTHER_NAME	■ USEREVENT_SCRIPT
■ DESCRIPTION_ITEM	■ OTHER_NAME_CATEGORY	■ VENDOR
■ DISCOUNT_ITEM	■ PARTNER	■ VENDOR_BILL
■ DOWNLOAD_ITEM	■ PARTNER_CATEGORY	■ VENDOR_CATEGORY
■ EMPLOYEE	■ PAYCHECK	■ VENDOR_CREDIT
■ END_TO_END_TIME	■ PAYCHECK_JOURNAL	■ VENDOR_PAYMENT
■ ENTITY		■ VENDOR_RETURN_AUTHORIZATION
■ ENTITY_ACCOUNT_MAPPING		■ WEBSITE
■ ESTIMATE		■ WORKFLOW_ACTION_SCRIPT

<ul style="list-style-type: none"> EXPENSE_CATEGORY EXPENSE_REPORT FAIR_VALUE_PRICE 	<ul style="list-style-type: none"> PAYMENT_EVENT PAYMENT_ITEM PAYMENT_METHOD PAYROLL_ITEM PCT_COMPLETE_PROJECT_REVENUE_RULE PERIOD_END_JOURNAL PERMISSION PHONE_CALL PORTLET PRICE_BOOK PRICE_LEVEL PRICING_GROUP PROJECT_EXPENSE_TYPE PROJECT_TASK 	<ul style="list-style-type: none"> WORK_ORDER WORK_ORDER_CLOSE WORK_ORDER_COMPLETION WORK_ORDER_ISSUE WORKPLACE
--	---	--

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/search Module Script Samples](#).

```
//Add additional code
...
var mySearch = search.create({
    type: search.Type.CUSTOMER,
    filters: filters,
    columns: columns
});
...
//Add additional code
```

N/sftp Module

The sftp module provides a way to upload and download files from external SFTP servers.

SFTP servers can be hosted by your organization or by a third party. NetSuite does not provide SFTP server functionality.

All SFTP transfers to or from NetSuite must originate from SuiteScript. It is not possible for external clients to initiate file transfers using SFTP.

Note: To use an external server to initiate a NetSuite file transfer that doesn't use SFTP, you can use RESTlets or SuiteTalk (Web Services). In SuiteScript, RESTlets can respond to requests containing file data and save them in the File Cabinet. RESTlets can also respond to requests for file data by loading the contents from the File Cabinet and returning them in the response. Note that binary file content must be received or sent as Base64 encoded Strings. See [SuiteScript 2.0 RESTlet Script Type](#) for more information.

In SuiteTalk, applications can invoke CRUD operations on the File Record to populate or change the contents of the File Cabinet. See the help topics [SuiteTalk \(Web Services\) Platform Guide](#) and [File](#) for more information.

- [N/sftp Module Members](#)
- [Connection Object Members](#)
- [N/sftp Module Script Sample](#)
- [Setting up an SFTP Transfer](#)
- [SFTP Authentication](#)
- [Supported Cipher Suites and Host Key Types](#)
- [Supported SuiteScript File Types](#)

N/sftp Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	sftp.Connection	Object	Server-side scripts	Represents a connection to the account on the remote FTP server.
Method	sftp.createConnection(options)	sftp.Connection	Server-side scripts	Establishes a connection to a remote FTP server.

Connection Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Connection.download(options)	file.File	Server-side scripts	Downloads a file from the remote FTP server
	Connection.upload(options)	void	Server-side scripts	Uploads a file to the remote FTP server.

N/sftp Module Script Sample

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

Important: Before you run this script, you must replace the GUID and host key with one specific to your account. The user name, URL, and directory values in this sample are also placeholders. Before using this sample, replace the placeholder values with valid values from your NetSuite account.

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following example uploads and downloads 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 [Form.addCredentialField\(options\)](#). Also see [N/https Module Script Sample](#) for a Suitelet example that shows creating a form field that generates a GUID.

```
/**
 * @NApiVersion 2.x
 */
require(['N/sftp', 'N/file'],
function(sftp, file) {
    var myPwdGuid = "B34672495064525E5D65032D63B52301";
    var myHostKey = "AAA1234567890Q=";

    // establish connection to remote FTP server

    var connection = sftp.createConnection({
        username: 'myuser',
        passwordGuid: myPwdGuid, // references var myPwdGuid
        url: 'host.somewhere.com',
        directory: 'myuser/wheres/my/file',
        hostKey: myHostKey // references var myHostKey
    });

    // specify the file to upload using the N/file module

    var myFileToUpload = file.create({
        name: 'originalname.js',
        fileType: file.fileType.PLAINTEXT,
        contents: 'I am a test file. Hear me roar.'
    });

    // 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

var downloadedFile = connection.download({
    directory: 'relative/path/to/file',
    filename: 'downloadMe.js'
});
});

```

Setting up an SFTP Transfer

- [Development Preparation for SFTP transfers](#)
- [Execution of an SFTP transfer](#)

Development Preparation for SFTP transfers

To successfully connect to your SFTP server with SuiteScript, the following steps are recommended:

1. Talk to your SFTP service provider about your plans.
 - Determine the connection properties required to connect with your external SFTP server. For example:
 - username
 - password
 - url
 - port
 - upload/download directories
 - host key
 - host key type
 - Make sure that you know your provider's practices around host key changes, maintenance and failover. For example, find out if there are multiple URLs or ports to try.
 - Check compatibility with the SFTP ciphers supported by NetSuite. See [Supported Cipher Suites and Host Key Types](#).
 - Determine if your provider requires at-rest file encryption (in addition to what the SFTP protocol provides during transfer). Decide if you need to add file encryption.
2. Build a credential management Suitelet to capture username and password token. Then, test the connection.
 - Create custom fields to store the user's SFTP username and password token
 - Implement the Suitelet.
 - a. Draw a form on a GET request.
 - b. Save the username and password token on a POST request.
 - c. Test the connection.

See [Creating a Suitelet Form that Contains a Credential Field](#).

- Build a server-side script to handle operations such as:
 - Load a File Cabinet file and upload it to the SFTP server.
 - Download an on demand file from the SFTP server and save it in File Cabinet.

Execution of an SFTP transfer

The following steps occur during a successful SFTP transfer using SuiteScript:

1. User submits their SFTP credentials via a Suitelet.
2. Suitelet captures and stores the credential token.
3. A server-side script is triggered.
4. Script identifies the appropriate credential token and other connection attributes, and establishes the SFTP connection.
5. Script requests the transfer.

SFTP Authentication

Please review the following sections for an overview of SFTP authentication when using SuiteScript.

- [Credential Tokenization](#)
- [Creating a Suitelet Form that Contains a Credential Field](#)
- [Reading the Credential Token in a Suitelet](#)
- [Credential Management](#)
- [Credential GUID Persistence](#)
- [Protocols](#)
- [Host Key Verification](#)
- [Retrieving the Host Key of an External SFTP Server](#)

Only username/password based authentication is supported. Public key based authentication is not supported.

Credential Tokenization

SuiteScript provides the ability for users to securely store authentication credentials in such a way that scripts are able to utilize encrypted saved credentials without being able to see their contents. The script author must specify which scripts and domains are permitted for use with the credential. To restrict the credential for use by SuiteScript automation triggered by the same user who originally saved the credential, the script author can set the `restrictToCurrentUser` parameter.

Creating a Suitelet Form that Contains a Credential Field

 **Note:** Credential fields have a default maximum length of 32 characters. If needed, use the `Field.maxLength` property to change this value

```
...
if(request.method === context.Method.GET){
    var form = ui.createForm({title: 'Enter SFTP Credentials'});
    var credField = form.addCredentialField({
        id: 'custfield_sftp_password_token',
        label: 'SFTP Password',
        restrictToScriptIds: ['customscript_sftp_script'],
        restrictToDomains: ['acmebank.com'],
        restrictToCurrentUser: true //Depends on use case
    });
    credField.maxLength = 64;
```



```

form.addSubmitButton();
response.writePage(form);
}
...

```

Reading the Credential Token in a Suitelet

Note that the following code snippet is not a fully functional sample.

```

...
var request = context.request;
if(request.method === context.Method.POST){
    // Read the request parameter matching the field ID we specified in the form
    var passwordToken = request.parameters.custfield_sftp_password_token;
    log.debug({
        title: 'New password token',
        details: passwordToken
    });
    // In a real-world script, "passwordToken" is saved into a custom field here...
}
...

```

Credential Management

User passwords can be stored using secure Credential Fields. This type of field is available on the [serverWidget.Form](#) Object in the [N/ui/serverWidget Module](#).

Encrypted *custom* fields do not support tokenization and are not compatible with the SFTP module. Instead, you can add a credential field using [Form.addCredentialField\(options\)](#).

Credential GUID Persistence

Scripts may store credential tokens as convenient for the script author. Credential tokens are not related to the password in its original or encrypted form within NetSuite. These tokens are unique identifiers which allow a script to refer to an encrypted secret securely stored within the SuiteCloud platform. Automatic password expiration is not currently provided, nor is it possible to view an inventory of saved credentials in the user interface.

Protocols

The SFTP module allows scripts to transfer files using the SSH File Transfer Protocol only. Other file-based protocols such as FTP, FTPS, SCP are not supported by this module.

Host Key Verification

An SFTP server identifies itself using a host key when a client attempts to establish a connection. Host keys are unique keys that the underlying SSH protocol uses to allow the server to provide a fingerprint. Clients can verify that the expected server has responded to the connection request for a particular URL and port number.

SuiteScript requires that the host key is provided by the script attempting to connect so that the SFTP module can check the identity of the SFTP server. This security best practice is commonly referred to as "Strict Host Key Checking".

Host keys are used to verify the identity of the server, not the client. SFTP/SSH host keys have no relationship to SFTP key based authentication, which is not currently supported.

By design, there is no SuiteScript API call for checking the host key of a remote SFTP server, or an option to disable strict host key checking. The script must always know the host key ahead of time.

We recommend using OpenSSH's `ssh-keyscan` tool to check the host key of an external SFTP site. See [Retrieving the Host Key of an External SFTP Server](#) and [The OpenBSD's ssh-keyscan page](#).

Retrieving the Host Key of an External SFTP Server

An example usage checking the RSA host key of URL: `acme.com` at port: `1234` from a `*nix` shell follows:

```
$ ssh-keyscan -t rsa -p 1234 acme.com
AAATpn1P9jB+cQx9Jg9UeZjA1245X7SBDcRiKh+Sok56VzSw==
```

It is recommended to always pass the key type and port number. This practice helps to avoid ambiguity in the response from the external SFTP server.

Supported Cipher Suites and Host Key Types

SFTP connections are encrypted. For security reasons, NetSuite requires that the server to which a connection request is being made supports at least one of the following ciphers `aes128-ctr`, `aes192-ctr` or `aes256-ctr`. The preceding cipher specs refer to the AES cipher in Counter stream cipher mode using 128, 192 or 256 bit key sizes.

To check interoperability of your SFTP server or service provider, refer to the following table:

Communication protocol	<p>SFTP (SSH + FTP) is supported.</p> <p>Only CTR (and not CBC) ciphers are allowed. Your SFTP server can use the following encryption algorithms:</p> <ul style="list-style-type: none"> ■ AES 128-CTR ■ AES 192-CTR ■ AES 256-CTR <p>Files are not additionally encrypted during transfer. The entire transmission is encrypted by the SSH protocol.</p>
Authentication mechanism	Username/password only (not key-based)
SSH host key	With each connection request, you must supply the host key. Any host key changes need to be managed manually.
Guid	<p>The password guid should be a value generated by a credential field from a Suitelet using <code>Form.addCredentialField(options)</code>.</p> <p>The password guid field's originating credential field must include the SFTP domain on the <code>restrictToDomains</code> parameter.</p> <p>The password guid field's originating credential field must include the script utilizing the password guid on the <code>restrictToScriptIds</code> parameter.</p>

Firewall policy is at the discretion of your SFTP service provider.

Supported SuiteScript File Types

SuiteScript has two types of file objects: previously existing files in the NetSuite File Cabinet, and on demand files created using SuiteScript API calls such as `file.create(options)` or `Connection.download(options)`.

File Cabinet and on demand files are supported by [Connection.upload\(options\)](#).

Note that [Connection.download\(options\)](#) returns an on demand file object. For an on demand file to be saved into the File Cabinet, it must receive a folder id and be explicitly saved.

```
...
var downloadedFile = sftp.download({...});
downloadedFile.folder = 1234;
downloadedFile.save();
...
```



Important: It's possible that a file you are downloading may be encrypted, or your SFTP provider may expect an uploaded file in a encrypted format in accordance with that provider's security practices. Make sure that you understand your provider's expectations and the cryptographic capabilities in SuiteScript (see [N/crypto Module](#)).

Annotated Syntax Sample

```
require(['N/ftp', 'N/file'],
function (sftp, file)
{
    var connection = sftp.createConnection({
        /*
         * The Username supplied by the administrator of the external SFTP server.
         */
        username: 'myuser',
        /*
         * Refers to the Password supplied by the administrator of the external SFTP server.
         *
         * The Password Token/GUID obtained by reading the form POST parameter associated
         * with user submission of a form containing a Credential Field.
         *
         * Value would typically be read from a custom field.
         */
        passwordGuid: "B34672495064525E5D65032D63B52301",
        /*
         * The URL supplied by the administrator of the external SFTP server.
         */
        url: 'host.somewhere.com',
        /*
         * The SFTP Port number supplied by the administrator of the external SFTP server (defaults to 22).
         */
        port: 22,
        /*
         * The transfer directory supplied by the administrator of the external SFTP server (optional).
         */
        directory: 'transferfiles',
        /*
         * RSA Host Key obtained via ssh-keyscan tool.
         *
         * $ ssh-keyscan -t rsa -p 22 host.somewhere.com
         * AATpnlP9jB+cQx9Jq9UeZjAl245X7SBDcRiKh+Sok56VzSw==
         */
    });
});
```

```

    */
    hostKey: "AATpnlP9jB+cQx9Jq9UeZjAl245X7SBDcRiKh+Sok56VzSw=="
  });

  /*
   * Creating a simple file.
   */
  var myFileToUpload = file.create({
    name: 'originalname.js',
    fileType: file.fileType.PLAINTEXT,
    contents: 'I am a test file. Hear me roar.'
  });

  /*
   * Uploading the file to the external SFTP server.
   */
  connection.upload({
    /*
     * Subdirectory within the transfer directory specified when connecting (optional).
     */
    directory: 'relative/path/to/remote/dir',
    /*
     * Alternate file name to use instead of the one given to the file object (optional).
     */
    filename: 'newFileNameOnServer.js',
    /*
     * The file to upload.
     */
    file: myFileToUpload,
    /*
     * If a file already exists with that name, replace it instead of failing the upload.
     */
    replaceExisting: true
  });

  var downloadedFile = connection.download({
    /*
     * Subdirectory within the transfer directory specified when connecting (optional).
     */
    directory: 'relative/path/to/file',
    /*
     * The name of the file within the above directory on the external SFTP server which to download.
     */
    filename: 'downloadMe.js'
  });
});

```

sftp.Connection

Object Description	Represents a connection to the account on the remote FTP server.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .

Module	N/sftp Module
Since	2016.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/sftp Module Script Sample](#).

```
//Add additional code ...
// establish connection to the FTP server
var objConnection = sftp.createConnection({
    username: 'username',
    passwordGuid: pwdGuid,
    url: 'host.somewhere.com',
    directory: 'username/wheres/my/file'
    hostKey: myHostKey
});
...
//Add additional code
```

Connection.download(options)

Method Description	Downloads a file from the remote FTP server
Returns	file.File Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	100
Module	N/sftp Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.filename	string	Required	The name of the file to download.	2016.2
options.directory	string	Optional	The relative path to the directory that contains the file to download. By default, the path is set to the current directory. <div> Important: This input must take the form of a relative path.</div>	2016.2
options.timeout	number	Optional	The number of seconds to allow for the file to download.	2016.2

Parameter	Type	Required / Optional	Description	Since
			By default, this value is set to 300 seconds.	

Errors

Error Code	Thrown If
FTP_MAXIMUM_FILE_SIZE_EXCEEDED	The file size is greater than the maximum file size allowed by NetSuite.
FTP_INVALID_DIRECTORY	The directory does not exist on the remote FTP server.
FTP_TRANSFER_TIMEOUT_EXCEEDED	The transfer is taking longer than the specified options.timeout value.
FTP_INVALID_TRANSFER_TIMEOUT	The options.timeout value is either a negative value, zero or greater than 300 seconds.
FTP_FILE_DOES_NOT_EXIST	The options.filename does not exist in the options.directory location.
FTP_PERMISSION_DENIED	Access to the file or directory on the remote FTP server was denied.
CONNECTION_RESET	The connection was reset.
THE_REMOTE_PATH_FOR_FILE_IS_NOT_VALID	The file's remote path is invalid.
CONNECTION_CLOSED_BY_HOST	The connection was closed by the host.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/sftp Module Script Sample](#).

```
//Add additional code
...
var downloadedFile = objConnection.download({
    directory: 'relative/path/to/file',
    filename: 'downloadMe.js'
});
...
//Add additional code
```

Connection.upload(options)

Method Description	Uploads a file to the remote FTP server.
Returns	void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	100
Module	N/sftp Module

Since	2016.2
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.file	file.File	Required	The file to upload.	2016.2
options.filename	string	Optional	The name to give the uploaded file on server. By default, the filename is the same specified by options.file.  Note: Illegal characters are automatically escaped.	2016.2
options.directory	string	Optional	The relative path to the directory where the file should be upload to. By default, the path is set to the current directory.  Important: This input must take the form of a relative path.	2016.2
options.timeout	number	Optional	The number of seconds to allow for the file to upload. By default, this value is set to 300 seconds.  Important: This parameter does not specify the overall timeout limit. The value is only applied when no data is received within the specified period.	2016.2
options.replaceExisting	boolean true false	Optional	Indicates whether the file being uploaded should overwrite any file with the name options.filename that already exists in options.directory. If false, the FTP_FILE_ALREADY_EXISTS exception is thrown when a file with the same name already exists in the options.directory. By default, this value is false.	2016.2

Errors

Error Code	Thrown If
FTP_INVALID_DIRECTORY	The directory does not exist on the remote FTP server.
FTP_TRANSFER_TIMEOUT_EXCEEDED	The transfer is taking longer than the specified options.timeout value.
FTP_INVALID_TRANSFER_TIMEOUT	The options.timeout value is either a negative value, zero or greater than 300 seconds.
FTP_FILE_ALREADY_EXISTS	The options.replaceExisting value is false and a file with the same name exists in the remote directory.

Error Code	Thrown If
CONNECTION_RESET	The connection was reset.
THE_REMOTE_PATH_FOR_FILE_IS_NOT_VALID	The file's remote path is invalid.
CONNECTION_CLOSED_BY_HOST	The connection was closed by the host.
FTP_PERMISSION_DENIED	Access to the file or directory on the remote FTP server was denied.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/sftp Module Script Sample](#).

```
//Add additional code ...
objConnection.upload({
  directory: 'relative/path/to/remote/dir',
  filename: 'newFileNameOnServer',
  file: myFileToUpload,
  replaceExisting: true
});
...
//Add additional code
```

sftp.createConnection(options)

Method Description	Establishes a connection to a remote FTP server. To generate the <code>passwordguid</code> , you can create a suitelet that uses Form.addCredentialField(options) . Use the N/https Module to fetch the GUID value returned from the Suitelet's credential field. For more information, see Setting up an SFTP Transfer and Supported Cipher Suites and Host Key Types .
Returns	sftp.Connection , representing that connection.
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/sftp Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.url</code>	string	Required	The host of the remote account.	2016.2
<code>options.passwordGuid</code>	string	Required	The password GUID for the remote account.	2016.2

Parameter	Type	Required / Optional	Description	Since
<code>options.hostKey</code>	string	Required	The host key for the trusted fingerprint on the server.	2016.2
<code>options.username</code>	string	Optional	The username of the remote account. By default, the login is anonymous.	2016.2
<code>options.port</code>	number	Optional	The port used to connect to the remote account. By default, port 22 is used.	2016.2
<code>options.directory</code>	string	Optional	The remote directory of the connection. <div>  Note: The directory property is required if you use a remote server cannot resolve relative paths. </div>	2016.2
<code>options.timeout</code>	number	Optional	The number of seconds to allow for an established connection. By default, this value is set to 20 seconds.	2016.2
<code>options.hostKeyType</code>	string	Optional	The type of host key specified by <code>options.hostKey</code> . This value can be set to one of the following options: <ul style="list-style-type: none"> ■ dsa ■ ecdsa ■ rsa 	2016.2

Errors

Error Code	Thrown If
FTP_UNKNOWN_HOST	The host could not be found.
FTP_CONNECT_TIMEOUT_EXCEEDED	A connection could not be established within <code>options.timeout</code> seconds.
FTP_CANNOT_ESTABLISH_CONNECTION	The password/username was invalid or permission to access the directory was denied.
FTP_INVALID_PORT_NUMBER	The port number is invalid.
FTP_INVALID_CONNECTION_TIMEOUT	The <code>options.timeout</code> value is either a negative value, zero, or greater than 20 seconds.
FTP_INVALID_DIRECTORY	The directory does not exist on the remote FTP server.
FTP_INCORRECT_HOST_KEY	The host key does not match the presented host key on the remote FTP server.
FTP_INCORRECT_HOST_KEY_TYPE	The host key type and provided host key type do not match.
FTP_MALFORMED_HOST_KEY	The host key is not in the correct format. (e.g. base 64, 96+ bytes)

Error Code	Thrown If
FTP_PERMISSION_DENIED	Access to the file or directory on the remote FTP server was denied.
FTP_UNSUPPORTED_ENCRYPTION_ALGORITHM	The remote FTP server does not support one of NetSuite's approved algorithms. (e.g. aes256-ctr, es192-ctr, es128-ctr)
AUTHENTICATION_FAIL_TOO_MANY_INCORRECT_AUTHENTICATION_ATTEMPTS	There are too many incorrect authentication attempts.
NO_ROUTE_TO_HOST_FOUND	No route to the host can be found.
CONNECTION_RESET	The connect was reset.
CONNECTION_CLOSED_BY_HOST	The connection was closed by the host.
THE_REMOTE_PATH_FOR_FILE_IS_NOT_VALID	The file's remote path is invalid.
SFTPCREDENTIAL_ENCODING_ERROR	There is an SFTP credential encoding error.
UNABLE_TO_GET_SFTP_SERVER_ADDRESS	The SFTP server address is unavailable.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/sftp Module Script Sample](#).

```
//Add additional code
...
// establish connection to the ftp server

var objConnection = sftp.createConnection({
    username: 'username',
    passwordGuid: pwdGuid, // references var pwdGuid
    url: 'host.somewhere.com',
    directory: 'username/wheres/my/file'
    hostKey: myHostKey // references var myHostKey
});
...
//Add additional code
```

N/sso Module

Use the sso module to generate outbound single sign-on (SuiteSignOn) tokens. For example, to create a reference to a SuiteSignOn record, or to integrate with an external application.

For more information about the SuiteSignOn feature, see the help topic [Outbound Single Sign-on \(SuiteSignOn\)](#).

- [N/sso Module Member](#)

N/sso Module Script Sample

N/sso Module Member

Member Type	Name	Return Type	Supported Script Types	Description
Method	<code>sso.generateSuiteSignOnToken(options)</code>	string	Portlet scripts, user event scripts, and Suitelets	Generates a new SuiteSignOn token for a user

N/sso Module Script Sample

The following sample script shows how you can use the sso module.

These sample scripts use the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).



Important: Before you run this script, you must replace the ID for the SuiteSignOn record with a value specific to your account. 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](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following example generates a new OAuth token for a user. The SuiteSignOn feature must be enabled.

```
/**
 * @NApiVersion 2.x
 */
require(['N/sso'],
function(sso) {
    function generateSSOToken() {
        var suiteSignOnRecordId = 1;
        var url = sso.generateSuiteSignOnToken(suiteSignOnRecordId);
    }
    generateSSOToken();
});
```

sso.generateSuiteSignOnToken(options)

Method Description	Method used to generate a new SuiteSignOn token for a user.
	<div>  Note: To use this method, Outbound Single Sign-on and web services must be enabled in your account. To enable these features, go to Setup > Company > Enable Features. On the SuiteCloud tab, in the Manage Authentication section, select the SuiteSignOn check box. In the SuiteTalk section, select the Web Services check box. Click Save. </div>
Returns	URL, OAuth token, and any integration variables as a string

Supported Script Types	Portlet scripts, user event scripts, and Suitelets For more information, see SuiteScript 2.0 Script Types .
Governance	20 units
Module	N/sso Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.suiteSignOnId	string	required	<p>The scriptId specified on the SuiteSignOn record. To see a list of IDs for SuiteSignOn records, go to the SuiteSignOn list page (Setup > Integration > SuiteSignOn Setup > Integration > SuiteSignOn).</p> <div>  Note: NetSuite recommends that you create a custom scriptId for each SuiteSignOn record to avoid naming conflicts should you decide use SuiteBundler to deploy your scripts into other accounts. </div>	2015.2

Errors

Error Code	Message	Thrown If
INVALID_SSO	Invalid SuiteSignOn reference: {1}. That SuiteSignOn object does not exist or has been marked as inactive.	<p>The suiteSignOnId input parameter is invalid or does not exist.</p> <div>  Note: The suiteSignOnId input parameter must be a scriptId and not a internal id. </div>
SSO_CONFIG_REQD	The SuiteSignOn object {1} is not configured for use with this script. You must specify the script as a connection point for this SuiteSignOn.	The suiteSignOnId input parameter is missing.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/sso Module Script Sample](#).

```
//Add additional code
...
var suiteSignOnRecordId = 1;
var url = sso.generateSuiteSignOnToken('customssol');
...
//Add additional code
```

N/task Module

Load the task module to create tasks and place them in the internal NetSuite scheduling or task queue. Use the task module to schedule scripts, run Map/Reduce scripts, import CSV files, merge duplicate records, and execute asynchronous workflows.

Each task type has its own corresponding object types. Use the methods available to each object type to configure, submit, and monitor the tasks.

 **Note:** Regardless of task type, tasks are always triggered asynchronously.

- N/task Module Members
- ScheduledScriptTask Object Members
- ScheduledScriptTaskStatus Object Members
- MapReduceScriptTask Object Members
- MapReduceScriptTaskStatus Object Members
- CsvImportTask Object Members
- CsvImportTaskStatus Object Members
- EntityDeduplicationTask Object Members
- EntityDeduplicationTaskStatus Object Members
- SearchTask Object Members
- SearchTaskStatus Object Members
- WorkflowTriggerTask Object Members
- WorkflowTriggerTaskStatus Object Members
- N/task Module Script Samples

N/task Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	<code>task.ScheduledScriptTask</code>	Object	Server-side scripts	Encapsulates all the properties of a scheduled script task in SuiteScript. Use this object to place a scheduled script deployment into the NetSuite scheduling queue.
	<code>task.ScheduledScriptTaskStatus</code>	Object	Server-side scripts	Encapsulates the status of a scheduled script placed into the NetSuite scheduling queue.
	<code>task.MapReduceScriptTask</code>	Object	Server-side scripts	Encapsulates a map/reduce script deployment.
	<code>task.MapReduceScriptTaskStatus</code>	Object	Server-side scripts	Encapsulates the status of a map/reduce script deployment that has been submitted for processing.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	task.CsvImportTask	Object	Server-side scripts	Encapsulates the properties of a CSV import task. Use the methods and properties for this object to submit a CSV import task into the task queue and asynchronously import record data into NetSuite.
	task.CsvImportTaskStatus	Object	Server-side scripts	Encapsulates the status of a CSV import task placed into the NetSuite scheduling queue.
	task.EntityDeduplicationTask	Object	Server-side scripts	Encapsulates all the properties of a merge duplicate records task request. Use the methods and properties of this object to submit a merge duplicate record job task into the NetSuite task queue.
	task.EntityDeduplicationTaskStatus	Object	Server-side scripts	Encapsulates the status of a merge duplicate record task placed into the NetSuite task queue.
	task.SearchTask	Object	Server-side scripts	Encapsulates the properties required to initiate an asynchronous search.
	task.SearchTaskStatus	Object	Server-side scripts	Encapsulates the status of an asynchronous search initiation task that is placed into the NetSuite task queue.
	task.WorkflowTriggerTask	Object	Server-side scripts	Encapsulates all the properties required to asynchronously initiate a workflow. Use WorkflowTriggerTask to create a task that initiates an instance of a specific workflow.
	task.WorkflowTriggerTaskStatus	Object	Server-side scripts	Encapsulates the status of an asynchronous workflow initiation task placed into the NetSuite task queue.
Method	task.create(options)	task.ScheduledScriptTask task.MapReduceScriptTask task.CsvImportTask task.EntityDeduplicationTask task.SearchTask task.WorkflowTriggerTask	Server-side scripts	Creates an object for a specific task type and returns the task object.
	task.checkStatus(options)	task.ScheduledScriptTaskStatus task.MapReduceScriptTaskStatus task.CsvImportTaskStatus task.EntityDeduplicationTaskStatus task.SearchTaskStatus task.WorkflowTriggerTaskStatus	Server-side scripts	Returns a task status object associated with a specific task ID.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Enum	<code>task.TaskType</code>	enum	Server-side scripts	Enumeration that holds the string values for the types of task objects, supported by the N/task Module , that you can create with task.create(options) .
	<code>task.TaskStatus</code>	enum	Server-side scripts	Enumeration that holds the string values for the possible status of tasks created and submitted with the N/task Module .
	<code>task.MasterSelectionMode</code>	enum	Server-side scripts	Enumeration that holds the string values for supported master selection modes when merging duplicate records with task.EntityDeduplicationTask .
	<code>task.DedupeMode</code>	enum	Server-side scripts	Enumeration that holds the string values for available deduplication modes when merging duplicate records with task.EntityDeduplicationTask .
	<code>task.DedupeEntityType</code>	enum	Server-side scripts	Enumeration that holds the string values for entity types for which you can merge duplicate records with task.EntityDeduplicationTask .
	<code>task.MapReduceStage</code>	enum	Server-side scripts	Enumeration that holds the string values for the stages of a map/reduce script deployment, which is encapsulated by the task.MapReduceScriptTask object.

ScheduledScriptTask Object Members

The following members are called on [task.ScheduledScriptTask](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>ScheduledScriptTask.submit()</code>	string	Server-side scripts	Directs NetSuite to place a scheduled script deployment into the NetSuite scheduling queue and returns a unique ID for the task.
Property	<code>ScheduledScriptTask.scriptId</code>	number string	Server-side scripts	Internal ID (as a number), or script ID (as a string) for the script record associated with a task.ScheduledScriptTask object.
	<code>ScheduledScriptTask.deploymentId</code>	number string	Server-side scripts	Internal ID (as a number), or script ID (as a string), for the script

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				deployment record associated with a task.ScheduledScriptTask object.
	ScheduledScriptTask.params	Object	Server-side scripts	Object with key/value pairs that override the static script parameter field values on the script deployment.

ScheduledScriptTaskStatus Object Members

The following members are called on [task.ScheduledScriptTaskStatus](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	ScheduledScriptTaskStatus.scriptId	read-only number	Server-side scripts	Internal ID for a script record associated with a specific task.ScheduledScriptTask object.
	ScheduledScriptTaskStatus.deployId	read-only number	Server-side scripts	Internal ID for a script deployment record associated with a specific task.ScheduledScriptTask object.
	ScheduledScriptTaskStatus.status	task.TaskStatus	Server-side scripts	Status for a scheduled script task. Returns a task.TaskStatus enum value.

MapReduceScriptTask Object Members

The following members are called on [task.MapReduceScriptTask](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	MapReduceScriptTask.submit()	string	Server-side scripts	Submits a map/reduce script deployment for processing.
Property	MapReduceScriptTask.scriptId	number string	Server-side scripts	Internal ID (as a number), or script ID (as a string), for the map/reduce script record.
	MapReduceScriptTask.deployId	number string	Server-side scripts	Internal ID (as a number), or script ID (as a string), for the script deployment record for a map/reduce script.
	MapReduceScriptTask.params	Object	Server-side scripts	Object that represents key/value pairs that override static script parameter field values on the script deployment record.

MapReduceScriptTaskStatus Object Members

The following members are called on the [task.MapReduceScriptTaskStatus](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	MapReduceScriptTaskStatus.getPercentageCompleted()	number	Server-side scripts	Returns the current percentage complete for

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				the current stage of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingMapCount()	number	Server-side scripts	Returns the total number of records or rows not yet processed by the map stage of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getTotalMapCount()	number	Server-side scripts	Returns the total number of records or rows passed as input to the map stage of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingMapSize()	number	Server-side scripts	Returns the total number of bytes not yet processed by the map stage, as a component of total size, of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingReduceCount()	number	Server-side scripts	Returns the total number of records or rows not yet processed by the reduce stage of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getTotalReduceCount()	number	Server-side scripts	Returns the total number of record or row inputs to the reduce stage of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingReduceSize()	number	Server-side scripts	Returns the total number of bytes not yet processed by the reduce stage, as a component of total size, of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingOutputCount()	number	Server-side scripts	Returns the total number of records or rows not yet processed by a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getPendingOutputSize()	number	Server-side scripts	Returns the total size in bytes of all key/value pairs written as output, as a component of total size, by a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getTotalOutputCount()	number	Server-side scripts	Returns the total number of records or rows passed as inputs to the output phase of a task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.getTotalSize()	number	Server-side scripts	Returns the total size in bytes of all stored work in progress by a task.MapReduceScriptTask .
Property	MapReduceScriptTaskStatus.scriptId	read-only number string	Server-side scripts	Internal ID for a map/reduce script record associated with a specific task.MapReduceScriptTask .
	MapReduceScriptTaskStatus.deployId	read-only number string	Server-side scripts	Internal ID for a script deployment record associated with a specific task.MapReduceScriptTask .

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>MapReduceScriptTaskStatus.status</code>	<code>task.TaskStatus</code>	Server-side scripts	Status for a map/reduce script task. Returns a <code>task.TaskStatus</code> enum value.
	<code>MapReduceScriptTaskStatus.stage</code>	<code>task.MapReduceStage</code>	Server-side scripts	The current stage of a map/reduce script deployment that is being processed. See <code>task.MapReduceStage</code> for supported values.

CsvImportTask Object Members

The following members are called on `task.CsvImportTask`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>CsvImportTask.submit()</code>	string	Server-side scripts	Directs NetSuite to place a CSV import task into the NetSuite task queue and returns a unique ID for the task.
Property	<code>CsvImportTask.importFile</code>	<code>file.File</code> string	Server-side scripts	CSV file to import. Use a <code>file.File</code> object or a string that represents the CSV text to be imported.
	<code>CsvImportTask.mappingId</code>	number string	Server-side scripts	Script ID or internal ID of the saved import map that you created when you ran the Import Assistant.
	<code>CsvImportTask.queueId</code>	number	Server-side scripts	Overrides the Queue Number property under Advanced Options on the Import Options page of the Import Assistant.
	<code>CsvImportTask.name</code>	string	Server-side scripts	Name for the CSV import task.
	<code>CsvImportTask.linkedFiles</code>	Object	Server-side scripts	A map of key/value pairs that sets the data to be imported in a linked file for a multi-file import job, by referencing a file in the file cabinet or the raw CSV data to import.

CsvImportTaskStatus Object Members

The following members are called on `task.CsvImportTaskStatus`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>CsvImportTaskStatus.status</code>	<code>task.TaskStatus</code>	Server-side scripts	Status for a CSV import task. Returns a <code>task.TaskStatus</code> enum value.

EntityDeduplicationTask Object Members

The following members are called on `task.EntityDeduplicationTask`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>EntityDeduplicationTask.submit()</code>	string	Server-side scripts	Directs NetSuite to place the merge duplicate records task into the NetSuite task queue and returns a unique ID for the task.
Property	<code>EntityDeduplicationTask.entityType</code>	<code>task.DedupeEntityType</code>	Server-side scripts	Sets the type of entity on which you want to merge duplicate records.
	<code>EntityDeduplicationTask.masterRecordId</code>	number	Server-side scripts	When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record.
	<code>EntityDeduplicationTask.masterSelectionMode</code>	<code>task.MasterSelectionMode</code>	Server-side scripts	When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record.
	<code>EntityDeduplicationTask.dedupeMode</code>	<code>task.DedupeMode</code>	Server-side scripts	Sets the mode in which to merge or delete duplicate records.
	<code>EntityDeduplicationTask.recordIds</code>	number[]	Server-side scripts	Number array of record internal IDs to perform the merge or delete operation on.

EntityDeduplicationTaskStatus Object Members

The following members are called on `task.EntityDeduplicationTaskStatus`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>EntityDeduplicationTaskStatus.status</code>	<code>task.TaskStatus</code>	Server-side scripts	Status for a merge duplicate record task.

SearchTask Object Members

The following members are called on `task.SearchTask`.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	SearchTask.addInboundDependency()	void	Server-side scripts	Adds a scheduled script task or map/reduce script task to the search task as a dependent script. Dependent scripts are processed automatically when the search task is complete. For more information, see the help topic SuiteCloud Processors .
	SearchTask.submit()	string	Server-side scripts	Places the asynchronous search initiation task into the SuiteScript task queue, and returns a unique ID for the task.
Property	SearchTask.fileId	number	Server-side scripts	ID of the CSV file to export search results into.
	SearchTask.filePath	string	Server-side scripts	Path of the CSV file to export search results into.
	SearchTask.inboundDependencies	Object	Server-side scripts	Object that contains key/value pairs to describe the dependent scripts added to the search task.
	SearchTask.savedSearchId	number	Server-side scripts	ID of the saved search to be executed during the task.

SearchTaskStatus Object Members

The following members are called on [task.SearchTaskStatus](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	SearchTaskStatus.fileId	number	Server-side scripts	ID of the CSV file into which search results are exported.
	SearchTaskStatus.savedSearchId	number	Server-side scripts	ID of the saved search executed during the task.
	SearchTaskStatus.status	task.TaskStatus	Server-side scripts	Status of an asynchronous search task placed in the NetSuite task queue.
	SearchTaskStatus.taskId	number	Server-side scripts	ID of the asynchronous task.

WorkflowTriggerTask Object Members

The following members are called on [task.WorkflowTriggerTask](#).

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	WorkflowTriggerTask.submit()	string	Server-side scripts	Directs NetSuite to place the asynchronous workflow initiation task into the NetSuite scheduling

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				queue and returns a unique ID for the task.
Property	<code>WorkflowTriggerTask.recordType</code>	string	Server-side scripts	Record type of the workflow base record. For example, customer, salesorder, or lead.
	<code>WorkflowTriggerTask.recordId</code>	number	Server-side scripts	Internal ID of the workflow definition base record. For example, 55 or 124.
	<code>WorkflowTriggerTask.workflowId</code>	number string	Server-side scripts	Internal ID (as a number), or script ID (as a string), for the workflow definition.
	<code>WorkflowTriggerTask.params</code>	Object	Server-side scripts	Object that contains key/value pairs to set default values on fields specific to the workflow.

WorkflowTriggerTaskStatus Object Members

The following members are called on the `task.WorkflowTriggerTaskStatus` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>WorkflowTriggerTaskStatus.status</code>	<code>task.TaskStatus</code>	Server-side scripts	Status for a asynchronous workflow placed in the NetSuite task queue.

N/task Module Script Samples

Some of the following script samples use the `require` function so that you can copy the script into the debugger and test it. However, you must use the `define` function in your entry point script (the script you attach to a script record). For more information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Sample 1

The following script sample submits a map/reduce script deployment.

 **Important:** Before you use this script sample, you must create a map/reduce script file, upload the file to NetSuite, and create a script record and script deployment record for it. For help working with map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#). Additionally, you must edit the sample and replace all hard-coded IDs with values that are valid in your NetSuite account.

```
/**
 * @ApiVersion 2.x
 */
require(['N/task', 'N/runtime', 'N/email'],
function(task, runtime, email) {
    function submitMapReduceDeployment() {
```

```

// 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.
var mapReduceScriptId = 'customscript_test_mapreduce_script';
log.audit('mapreduce id: ', mapReduceScriptId);

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

// Submit the map/reduce task.
var 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.
var taskStatus = task.checkStatus(mrTaskId);
if (taskStatus.status === 'FAILED') {
    var authorId = -5;
    var recipientEmail = 'notify@myCompany.com';
    email.send({
        author: authorId,
        recipients: recipientEmail,
        subject: 'Failure executing map/reduce job!',
        body: 'Map reduce task: ' + mapReduceScriptId + ' has failed.'
    });
}

submitMapReduceDeployment();
});

```

Sample 2

The following script 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 script retrieves the task status using the task ID.

Note: Some of the values in this script 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.

```

/**
 *@NApiVersion 2.x
 */

```

```

require(['N/task'],
function(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 id, filters, and columns
    //   values as appropriate):
    //
    //   var mySearch = search.create({
    //       type: search.Type.SALES_ORDER,
    //       id: 'customsearch_my_search',
    //       filters: [...],
    //       columns: [...]
    //   });
    //   mySearch.save();
    //   var 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.
    var savedSearchId = -10;

    // Create the search task.
    var 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.
    var myTaskId = myTask.submit();

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

    // Optionally, create new variables to represent values used previously in
    // this script. You may want to use these variables in additional logic you
    // add to this script.
    var myFileId = taskStatus.fileId;
    var mySavedSearchId = taskStatus.savedSearchId;

    // 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.

    }
}

```

```
});
```

Sample 3

The following script 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](#).

Note: This script 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 snippets below. Before using this sample, create these parameters in the script record. For more information, see the help topic [Creating Script Parameters](#).

```
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';
    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 within a dependent scheduled script, consider the following script sample.

```
/**
 * @NApiVersion 2.x
 * @NScriptType ScheduledScript
 */
define(['N/file', 'N/log', 'N/email', 'N/runtime'],

// Load the search results file and send an email with the file attached and
// the number of rows in the file.
function(file, log, email, runtime) {
    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 within 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'],

    // Load the search results file, count the number of letters in the file, and
    // store this count in another file.
    function(runtime, file, log, email) {

        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
    };
});

```

task.ScheduledScriptTask

Object Description	<p>Encapsulates all the properties of scheduled script task in SuiteScript. Use this object to place a scheduled script deployment into the NetSuite scheduling queue.</p> <p>To use the <code>ScheduledScriptTask</code> Object:</p> <ol style="list-style-type: none"> 1. In the NetSuite UI, create the script record and script deployment record. 2. Use <code>task.create(options)</code> to create the <code>ScheduledScriptTask</code> object. 3. Use the <code>ScheduledScriptTask</code> object properties to set the script and deployment properties. 4. Use <code>ScheduledScriptTask.submit()</code> to deploy the scheduled script to the NetSuite scheduling queue. 5. Use the properties for the <code>task.ScheduledScriptTaskStatus</code> object to get the status of the scheduled script. <p>For a complete list of this object's methods and properties, see ScheduledScriptTask Object Members.</p> <p>For more information about scheduled scripts in NetSuite, see SuiteScript 2.0 Scheduled Script Type.</p>
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var scriptTask = task.create({taskType: task.TaskType.SCHEDULED_SCRIPT});
scriptTask.scriptId = 1234;
scriptTask.deploymentId = 'customdeploy1';
scriptTask.params = {searchId: 'custsearch_456'};
var scriptTaskId = scriptTask.submit();
...
//Add additional code

```

ScheduledScriptTask.submit()

Method Description	<p>Directs NetSuite to place a scheduled script deployment into the NetSuite scheduling queue and returns a unique ID for the task.</p> <p>Additionally, note the following:</p>
---------------------------	--

	<ul style="list-style-type: none"> ■ The scheduled script must have a status of Not Scheduled on the Script Deployment page. If the script status is set to Testing on the Script Deployment page, this method will not place the script into the scheduling queue. ■ If the deployment status on the Script Deployment page is set to Scheduled, the script will be placed into the queue according to the time(s) specified on the Script Deployment page. ■ Only administrators can run scheduled scripts. If a user event script calls <code>ScheduledScriptTask.submit()</code>, the user event script has to be deployed with admin permissions. ■ A scheduled script can be submitted for processing only if there is no unfinished scheduled script task for the same script ID and script deployment ID. Therefore, if a scheduled script resubmits itself, the actual resubmit does not occur until the current execution completes. This delay is necessary to avoid the existence of two unfinished tasks for the same deployment of the same script. For this reason, if a scheduled script uses the <code>submit()</code> method to resubmit itself, then at runtime, no task ID is returned when the scheduled script is submitted.
Returns	The task ID as a string, except as noted above.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	20 units
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Failed to submit job request: {reason}	Task cannot be submitted.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var scheduledScriptTaskId = scriptTask.submit();
...
//Add additional code
```

ScheduledScriptTask.scriptId

Property Description	Internal ID (as a number), or script ID (as a string), for the script record associated with a <code>task.ScheduledScriptTask</code> object.
Type	number string
Governance	20 units
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var scheduledScriptId = 34;
...
//Add additional code
```

ScheduledScriptTask.deploymentId

Property Description	Internal ID (as a number), or script ID (as a string), for the script deployment record associated with a task.ScheduledScriptTask Object.
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code .
..
scheduledTask.deploymentId = 1;
...
//Add additional code
```

ScheduledScriptTask.params

Property Description	Object with key/value pairs that override static script parameter field values on the script deployment. Use these parameters for the task.ScheduledScriptTask object to programmatically pass values to the script deployment. For more information about script parameters, see the help topic Creating Script Parameters Overview .
Type	object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
scriptTask.params = {searchId: 'custsearch_456'};
...
//Add additional code
```

task.ScheduledScriptTaskStatus

Object Description	Encapsulates the properties and status of a scheduled script placed into the NetSuite scheduling queue. Use task.checkStatus(options) with the unique ID for the scheduled script task to get the ScheduledScriptTaskStatus Object. For a complete list of this object's properties, see ScheduledScriptTaskStatus Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var res = task.checkStatus(scriptTaskId);
log.debug('Initial status: ' + res.status);
...
//Add additional code
```

ScheduledScriptTaskStatus.scriptId

Property Description	Internal ID for a script record associated with a specific task.ScheduledScriptTask Object. Use this ID to get more details about the script record for the scheduled task.
Type	read-only number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
log.audit('Initial status: ' + status.scriptId);
...
//Add additional code
```

ScheduledScriptTaskStatus.deploymentId

Property Description	Internal ID for a script deployment record associated with a specific task.ScheduledScriptTask Object. Use this ID to get more details about the script deployment record for the scheduled task.
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
log.audit({
  details: 'Deployment ID: ' + status.scriptId
});
```



```
...
//Add additional code
```

ScheduledScriptTaskStatus.status

Property Description	Status for a scheduled script task. Returns a task.TaskStatus enum value.
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
log.audit({
    details: 'Status: ' + summary.status
});
...
//Add additional code
```

task.MapReduceScriptTask

Object Description	<p>Encapsulates the properties of a map/reduce script deployment. You can use this object to programmatically submit a script deployment for processing. To use the <code>MapReduceScriptTask</code> object:</p> <ul style="list-style-type: none"> ■ In the NetSuite UI, create the script record and script deployment records. ■ Use task.create(options) to create the <code>MapReduceScriptTask</code> object. ■ Use the <code>MapReduceScriptTask</code> object properties to set the script and deployment properties. ■ Use MapReduceScriptTask.submit() to submit the deployment for processing. ■ Use the properties for the task.MapReduceScriptTaskStatus object to get the status of the map/reduce script. <p>For a complete list of this object's methods and properties, see MapReduceScriptTask Object Members.</p> <p>For general information about map/reduce scripts, see Map/Reduce Key Concepts.</p>
---------------------------	---

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var mrTask = task.create({taskType: task.TaskType.MAP_REDUCE});
mrTask.scriptId = mapReduceScriptId;
mrTask.deploymentId = 1;
var mrTaskId = mrTask.submit();
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTask.submit()

Method Description	Submits a map/reduce script deployment for processing. For more information, see task.MapReduceScriptTask . Additionally, note that a map/reduce script can be submitted for processing only if there is no unfinished map/reduce script task for the same script ID and script deployment ID. For this reason, if a map/reduce script resubmits itself, the actual resubmit does not occur until the current execution completes. This delay is necessary to avoid the existence of two unfinished tasks for the same deployment of the same script. Therefore, if a map/reduce script uses the submit() method to resubmit itself, then at runtime, no task ID is returned when the map/reduce script is submitted. For general information about the execution of map/reduce scripts, see SuiteScript 2.0 Map/Reduce Script Submission .
Returns	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	20 units
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Failed to submit job request: {reason}	Task cannot be submitted.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var mrTaskId = mrTask.submit();
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTask.scriptId

Property Description	Internal ID (as a number), or script ID (as a string), for the map/reduce script record.
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var mapReduceScriptId = 34;
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTask.deploymentId

Property Description	Internal ID (as a number) or script ID (as a string), for the script deployment record for a map/reduce script.
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
mrTask.deploymentId = 1;
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTask.params

Property Description	Object that represents key/value pairs that override static script parameter field values on the script deployment record. Use these parameters on a task.MapReduceScriptTask object to programmatically pass values to the script deployment. For more information about script parameters, see the help topic Creating Script Parameters Overview .
Type	object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...

mrTask.params = {doSomething: true};
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

task.MapReduceScriptTaskStatus

Object Description	Encapsulates the status of a map/reduce script deployment that was submitted for processing. Use task.checkStatus(options) with the unique ID for the map/reduce script task to get the MapReduceScriptTaskStatus object.
---------------------------	--

	For a complete list of this object's methods and properties, see MapReduceScriptTaskStatus Object Members . For general information about the execution of map/reduce scripts, see SuiteScript 2.0 Map/Reduce Script Submission .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
if (summary.stage === task.MapReduceStage.SUMMARIZE)
    log.audit('Almost done...');
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPercentageCompleted()

Method Description	Returns the current percentage complete for the current stage of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages . <div> Note: The input and summarize stages are either 0% or 100% complete at any time. </div>
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
```

```
...
var completion = taskStatus.getPercentageCompleted();
log.audit('Percentage Completed: ' + completion);
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingMapCount()

Method Description	Returns the total number of records or rows not yet processed by the map stage of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages .
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getPendingMapCount();
log.audit('Pending Map Count: ' + summary);
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getTotalMapCount()

Method Description	Returns the total number of records or rows passed as input to the map stage of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages .
---------------------------	---

Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getTotalMapCount();
log.audit('Total Map Count: ' + summary);
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingMapSize()

Method Description	Returns the total number of bytes not yet processed by the map stage, as a component of total size, of a <code>task.MapReduceScriptTask</code> . Use the <code>MapReduceScriptTaskStatus.stage</code> property to get the current stage. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages .
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	25 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getPendingMapSize();
```

```
log.audit('Pending Map Size: ' + summary);
...
//Add additional code
```

 **Note:** For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingReduceCount()

Method Description	Returns the total number of records or rows not yet processed by the reduce stage of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage. For general information about the reduce stage and other map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages .
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getPendingReduceCount();
log.audit({
    details: 'Pending Reduce Count: ' + summary
});
...
//Add additional code
```

 **Note:** For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getTotalReduceCount()

Method Description	Returns the total number of record or row inputs to the reduce stage of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages .
---------------------------	---

Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getTotalReduceCount();
log.audit({
    details: 'Reduce Count: ' + summary
});
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingReduceSize()

Method Description	Returns the total number of bytes not yet processed by the reduce stage, as a component of total size, of a task.MapReduceScriptTask . Use the MapReduceScriptTaskStatus.stage property to get the current stage.
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	25 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = taskStatus.getPendingReduceSize();
```

```
log.audit({
  details: 'Pending Reduce Size: ' + summary
});
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingOutputCount()

Method Description	Returns the total number of records or rows not yet processed by a task.MapReduceScriptTask .
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
var total = summary.getPendingOutputCount()
log.audit({
  title: 'Count',
  details: total
});
...
//Add additional code
```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getPendingOutputSize()

Method Description	Returns the total size in bytes of all key/value pairs written as output, as a component of total size, by a task.MapReduceScriptTask .
Returns	number

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	25 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
var total = summary.getPendingOutputSize()
log.audit({
    title: 'Size',
    details: total}
);
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getTotalOutputCount()

Method Description	Returns the total number of key/value pairs passed as inputs to the <code>SUMMARIZE</code> phase of a <code>task.MapReduceScriptTask</code> . Use the <code>MapReduceScriptTaskStatus.stage</code> property to get the current stage.
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	10 units
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
```

```

var total = summary.getTotalOutputCount()
log.audit({
  title: 'Total Entries Passed to Output',
  details: total
});
...
//Add additional code

```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.getCurrentTotalSize()

Method Description	Returns the total size in bytes of all stored work in progress by a <code>task.MapReduceScriptTask</code> .
Returns	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	25 units
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
var total = summary.getCurrentTotalSize()
log.audit({
  title: 'Size of Remaining Data to Process',
  details: total
});
...
//Add additional code

```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.scriptId

Property Description	Internal ID for a map/reduce script record associated with a specific <code>task.MapReduceScriptTask</code> .
-----------------------------	---

Type	read-only number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
log.audit({
    title: 'Script ID',
    details: summary.scriptId
});
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.deploymentId

Property Description	Internal ID for a script deployment record associated with a specific task.MapReduceScriptTask .
Type	read-only number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
log.audit({
    title: 'Deployment ID',
    details: summary.deploymentId
});
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.status

Property Description	Status of a map/reduce script deployment that was submitted for processing. Returns a task.TaskStatus enum value. For general details about the execution of map/reduce scripts, see SuiteScript 2.0 Map/Reduce Script Submission .
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
log.audit({
    title: 'Status',
```

```

        details: summary.status
    });

    ...
    //Add additional code

```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

MapReduceScriptTaskStatus.stage

Property Description	Current stage of processing for a map/reduce script deployment instance. See task.MapReduceStage for supported values. For general information about map/reduce stages, see Map/Reduce Key Concepts . For information about the execution of map/reduce scripts, see SuiteScript 2.0 Map/Reduce Script Submission .
Type	task.MapReduceStage
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
if (summary.stage === task.MapReduceStage.SUMMARIZE)
    log.audit({
        details: 'Almost done...'
    });
...
//Add additional code

```

Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

task.CsvImportTask

Object Description	Encapsulates the properties of a CSV import task. Use the methods and properties for this object to submit a CSV import task into the task queue and asynchronously import record data into NetSuite.
---------------------------	---

	<p>Use the <code>CsvImportTask</code> Object to perform the following types of tasks:</p> <ul style="list-style-type: none"> ■ Automate standard record data import for SuiteApp installations, demo environments, and testing environments. ■ Import data on a schedule using a scheduled script. ■ Build integrated CSV imports with RESTlets. <p>Use the following process to import CSV data with <code>CsvImportTask</code>:</p> <ul style="list-style-type: none"> ■ In the NetSuite UI, run the Import Assistant to set up the CSV mapping and import options. You must run the Import Assistant to set up the necessary mapping for the CSV import. You can use a sample file or files to set up the mapping. Note the following information: <ul style="list-style-type: none"> □ Script ID for import map. □ Any required linked files. For more information, see the help topic Importing CSV Files with the Import Assistant. ■ Use <code>task.create(options)</code> to create the <code>CsvImportTask</code> object. ■ Use the <code>CsvImportTask</code> object properties to set the script and deployment properties. ■ Use <code>CsvImportTask.submit()</code> to submit the import task to the NetSuite task queue. ■ Use the properties for the <code>task.CsvImportTaskStatus</code> object to get the status of the import process. <p>Use the following guidelines with the <code>CsvImportTask</code> Object:</p> <ul style="list-style-type: none"> ■ CSV imports performed within scripts are subject to the existing application limit of 25,000 records. ■ You cannot import data that is imported by (2-step) assistants in the UI, because these import types do not support saved import maps. This limitation applies to budget, single journal entry, single inventory worksheet, project tasks, and website redirects imports. ■ This object has access only to the field mappings of a saved import map; it does not have access to advanced import options defined in the Import Assistant, such as multi-threading and multiple queues. <p>Even if you set options to use multiple threads or queues for an import job and then save the import map, these settings are not available to <code>CsvImportTask</code>. When this object submits a CSV import job based on the saved import map, a single thread and single queue are used.</p> For a complete list of this object's methods and properties, see CsvImportTask Object Members.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var scriptTask = task.create({taskType: task.TaskType.CSV_IMPORT});
scriptTask.mappingId = 51;
var f = file.load('SuiteScripts/custjoblist.csv');
scriptTask.importFile = f;
scriptTask.linkedFiles = {'addressbook': 'street,city\nval1,val2', 'purchases': file.load('SuiteScripts/other.csv')};
var csvImportTaskId = scriptTask.submit();
...
```



```
//Add additional code
```

CsvImportTask.submit()

Method Description	Directs NetSuite to place a CSV import task into the NetSuite task queue and returns a unique ID for the task. Use CsvImportTaskStatus.status to view the status of a submitted task. This method throws errors resulting from inline validation of CSV file data before the import of data begins (the same validation that is performed between the mapping step and the save step in the Import Assistant). Any errors that occur during the import job are recorded in the CSV response file, as they are for imports initiated through the Import Assistant.
Returns	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	100 units
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Failed to submit job request: {reason}	Task cannot be submitted.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var csvImportTaskId = csvTask.submit();
...
//Add additional code
```

CsvImportTask.importFile

Property Description	CSV file to import. Use a file.File object or a string that represents the CSV text to be imported.
Type	file.File string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var f = file.load('SuiteScripts/custjoblist.csv');
scriptTask.importFile = f;
...
//Add additional code
```

CsvImportTask.mappingId

Property Description	Script ID or internal ID of the saved import map that you created when you ran the Import Assistant. See task.CsvImportTask .
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
scriptTask.mappingId = 51;
...
//Add additional code
```

CsvImportTask.queueId

Property Description	<p>Overrides the Queue Number property under Advanced Options on the Import Options page of the Import Assistant. Use this property to programmatically select an import queue and improve performance during the import.</p> <div> <p>Note: This property is only available if you have a SuiteCloud Plus license. For more information about using multiple queues when importing CSV files, see the help topics Queue Number and Use Multiple Threads and Multiple Queues to Run CSV Import Jobs.</p> </div>
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
scriptTask.queueId = 2;
...
//Add additional code
```

CsvImportTask.name

Property Description	Name for the CSV import task. You can optionally set a different name for a scripted import task. In the UI, this name appears on the CSV Import Job Status page.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
csvTask.name = 'Import Entities'
...
//Add additional code
```

CsvImportTask.linkedFiles

Property Description	A map of key/value pairs that sets the data to be imported in a linked file for a multi-file import job, by referencing a file in the file cabinet or the raw CSV data to import. The key is the internal ID of the record sublist for which data is being imported and the value is either a file.File object or the raw CSV data to import. You can assign multiple types of values to the <code>linkedFiles</code> property.
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
scriptTask.linkedFiles = {'addressbook': 'street,city\nval1,val2', 'purchases': file.load('SuiteScripts/other.csv')};
...
//Add additional code
```

task.CsvImportTaskStatus

Object Description	Encapsulates the status of a CSV import task placed into the NetSuite scheduling queue. Use task.checkStatus(options) with the unique ID for the CSV import task to get the <code>CsvImportTaskStatus</code> object. For a complete list of this object's properties, see CsvImportTaskStatus Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var csvTaskStatus = task.checkStatus({
    taskId: csvTaskId
});
if (csvTaskStatus.status === task.TaskStatus.FAILED)
...
//Add additional code
```

CsvImportTaskStatus.status

Property Description	Status for a CSV import task. Returns a task.TaskStatus enum value.
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module

Since	2015.2
-------	--------

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus({
    taskId: scriptTaskId
});
log.audit({
    title: 'Status',
    details: summary.status
});
...
//Add additional code
```

task.EntityDeduplicationTask

Object Description	<p>Encapsulates all the properties of a merge duplicate records task request. Use the methods and properties of this object to submit a merge duplicate record job task into the NetSuite task queue. When you submit a merge duplicate record task to NetSuite, SuiteScript enables you to use all of the same functionality available through the UI. Use SuiteScript to use the predefined duplicate detection rules, or you can define your own. After the records are merged or deleted, in the UI, the records no longer appear as duplicates at Lists > Mass Update > Entity Duplicate Resolution Lists > Mass Update > Entity Duplicate Resolution.</p> <p>For more information about merging duplicate records in NetSuite, see the help topic Merging or Deleting Duplicate Records.</p> <p>To use the <code>EntityDeduplicationTask</code> Object:</p> <ul style="list-style-type: none"> ■ Use <code>task.create(options)</code> to create the <code>EntityDeduplicationTask</code> object. ■ Use <code>EntityDeduplicationTask.entityType</code> to select the entity type on which you want to merge duplicate records. ■ Use <code>EntityDeduplicationTask.dedupeMode</code> to select the action to take for the duplicate records. ■ Use a <code>EntityDeduplicationTask.masterSelectionMode</code> enum value to identify which record to use as the master record in the merge. ■ If you use <code>MasterSelectionMode.SELECT_BY_ID</code> for the master selection mode, set the ID of the master record with <code>EntityDeduplicationTask.masterRecordId</code>. ■ Identify the duplicate records. Use the <code>search.duplicates(options)</code> method in the N/search Module to find the duplicate records. ■ Use <code>EntityDeduplicationTask.submit()</code> to submit the merge duplicate record task to the NetSuite task queue. ■ Use the properties for the <code>task.EntityDeduplicationTaskStatus</code> object to get the status of the merge duplicate record task.
---------------------------	---

	<p>Use the following guidelines with the <code>EntityDeduplicationTask</code> Object:</p> <ul style="list-style-type: none"> ■ You can only submit 200 records in a single merge duplicate records task. ■ The merge duplicate functionality on non-entity records is not supported in SuiteScript. ■ You must have full access to the Duplicate Record Management permission to merge duplicates. <p>For a complete list of this object's methods and properties, see EntityDeduplicationTask Object Members.</p>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var dedupeTask = task.create({taskType: task.TaskType.ENTITY_DEDUPLICATION});
dedupeTask.entityType = task.DedupeEntityType.CUSTOMER;
dedupeTask.dedupeMode = task.DedupeMode.MERGE;
dedupeTask.masterSelectionMode = task.MasterSelectionMode.MOST_RECENT_ACTIVITY;
dedupeTask.recordIds = ['107', '110'];
var dedupeTaskId = dedupeTask.submit();
...
//Add additional code
```

EntityDeduplicationTask.submit()

Method Description	Directs NetSuite to place the merge duplicate records task into the NetSuite task queue and returns a unique ID for the task. Use EntityDeduplicationTaskStatus.status to view the status of a submitted task.
Returns	task id as a string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	100 units
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Failed to submit job request: {reason}	Task cannot be submitted.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var dedupeTaskId = dedupeTask.submit();
...
//Add additional code
```

EntityDeduplicationTask.entityType

Property Description	<p>Sets the type of entity on which you want to merge duplicate records. Use a task.DedupeEntityType enum value to set the value.</p> <div> Note: If you set entityType to CUSTOMER, the system will automatically include prospects and leads in the task request. </div>
Type	task.DedupeEntityType
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.entityType = task.DedupeEntityType.CUSTOMER;
...
//Add additional code
```

EntityDeduplicationTask.masterRecordId

Property Description	<p>When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record. Use this property to set the ID of the master record that you want to use as the master record in the merge.</p> <div> Important: You must also select SELECT_BY_ID for the EntityDeduplicationTask.masterSelectionMode property, or NetSuite ignores this setting. </div>
Type	number

Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.masterSelectionMode = task.MasterSelectionMode.SELECT_BY_ID;
dedupeTask.masterRecordId = 107;
...
//Add additional code
```

EntityDeduplicationTask.masterSelectionMode

Property Description	When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record. Set this property to determine which of the duplicate records to keep or select the master record to use by ID. Use EntityDeduplicationTask.masterSelectionMode to set the value.
Type	task.MasterSelectionMode
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.masterSelectionMode = task.MasterSelectionMode.MOST_RECENT_ACTIVITY;
...
//Add additional code
```

EntityDeduplicationTask.dedupeMode

Property Description	Sets the mode in which to merge or delete duplicate records. Use a EntityDeduplicationTask.dedupeMode enum value to set the value.
-----------------------------	---

Type	task.DedupeMode
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.dedupeMode = task.DedupeMode.MERGE;
...
//Add additional code
```

EntityDeduplicationTask.recordIds

Property Description	Number array of record internal IDs to perform the merge or delete operation on. You can use the search.duplicates(options) method to identify duplicate records or create an array with record internal IDs.
Type	number[]
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.recordIds = ['107', '110'];
...
//Add additional code
```

task.EntityDeduplicationTaskStatus

Object Description	Encapsulates the status of a merge duplicate record task placed into the NetSuite task queue by EntityDeduplicationTask.submit() . Use task.checkStatus(options) with the unique ID for the merge duplicate records task to get this Object.
--------------------	--

	For a complete list of this object's properties, see EntityDeduplicationTaskStatus Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var dedupeTaskStatus = task.checkStatus({
    taskId: taskId
});
if (dedupeTaskStatus.status === task.TaskStatus.FAILED)
...
//Add additional code
```

EntityDeduplicationTaskStatus.status

Property Description	Status for a merge duplicate record task. Returns a task.TaskStatus enum value.
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus({
    taskId: scriptTaskId
});
log.audit({
    title: 'Status',
```

```

    details: summary.status
  });
  ...
  //Add additional code

```

task.SearchTask

Object Description	<p>Encapsulates the properties of a search task. Use the methods and properties for this object to submit a search task into the task queue, execute it asynchronously, and persist search results. Similar to SuiteAnalytics persisted search functionality, this capability is useful for searches across high volumes of data.</p> <p>You can create a <code>task.SearchTask</code> object using <code>task.create(options)</code>. Use the <code>task.SearchTask</code> object to do the following:</p> <ul style="list-style-type: none"> Set the search ID using the <code>SearchTask.savedSearchId</code> property. Set the file ID or file path of a CSV file in the File Cabinet. Search results are exported to this file. Use the <code>SearchTask.fileId</code> property or the <code>SearchTask.filePath</code> property. Exactly one of these properties must be set. If both are set, an error occurs. Add dependent scripts to the search task using <code>SearchTask.addInboundDependency()</code>. Dependent scripts are processed automatically when the search task is complete. Submit the search task to the NetSuite task queue using <code>SearchTask.submit()</code>. Get the status of a search task using the properties of the <code>task.SearchTaskStatus</code> object. <div>  Note: There a limit to the number of asynchronous searches running at the same time. The limit is set to be the same as the limit for CSV import. The file size limit is based on File Cabinet limits. </div>
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/task Module
Since	2017.1

Syntax

```

//Add additional code
...
var searchTask = task.create({
  taskType: task.TaskType.SEARCH
});
searchTask.savedSearchId = 51;
var path = 'ExportFolder/export.csv';
scriptTask.filePath = path;
var searchTaskId = searchTask.submit();
...
//Add additional code

```

SearchTask.addInboundDependency()

Method Description	<p>Adds a scheduled script task (<code>task.ScheduledScriptTask</code>) or map/reduce script task (<code>task.MapReduceScriptTask</code>) to the search task as a dependent script. Dependent scripts are</p>
---------------------------	---

	<p>processed automatically when the search task is complete. For more information, see the help topic SuiteCloud Processors.</p> <div>  Note: You can add only scheduled scripts or map/reduce scripts as dependent scripts to asynchronous search tasks. Other script types are not supported. </div> <p>When you use this method to add a dependent script, the script is considered an inbound dependency of the search task. The added script depends on the search task. For example, if you add a scheduled script task to a search task as a dependent script, the scheduled script depends on the search task. Because <code>addInboundDependency()</code> is called on the search task, any dependent scripts that you add are considered inbound dependencies.</p>
Returns	void
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/task Module
Since	2018.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
dependentScript	task.ScheduledScriptTask task.MapReduceScriptTask	Required	<p>The script to add as a dependent script to the search task.</p> <p>Use <code>task.create(options)</code> and the <code>task.TaskType</code> enum to create a script task with a type of <code>SCHEDULED_SCRIPT</code> or <code>MAP_REDUCE</code>. This script task is a task.ScheduledScriptTask object or a task.MapReduceScriptTask, and you can add this script task as a dependent script to the search task. The dependent script is processed when the search task is complete. You can add only one dependent script per call to <code>SearchTask.addInboundDependency()</code>.</p>	2018.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
// Add additional code
...
var scheduledScript = task.create({
    taskType: task.TaskType.SCHEDULED_SCRIPT
});
// Set the properties of the scheduled script task
scheduledScript.scriptId = 'customscript_as_ftr_ss';
...
```

```

var asyncTask = task.create({
    taskType: task.TaskType.SEARCH
});
// Set the properties of the search task
asyncTask.savedSearchId = 'customsearch35';
...

asyncTask.addInboundDependency(scheduledScript);

var asyncTaskId = asyncTask.submit();
...
// Add additional code

```

SearchTask.submit()

Method Description	Directs NetSuite to initiate the asynchronous search task and return a unique ID for the task. When the submission is successful, this method adds the internal IDs of any dependent scripts (added using SearchTask.addInboundDependency()) to the SearchTask.inboundDependencies property. Use task.SearchTaskStatus to view the status of a submitted task.
Returns	The task ID as a string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	100 units
Module	N/task Module
Since	2017.1

Errors

Error Code	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Task cannot be submitted.
SSS_MISSING_REQD_ARGUMENT	A required parameter is missing.
YOU_DO_NOT_HAVE_ACCESS_TO_THE_MEDIA_ITEM_YOU_SELECTED	You do not have permission to access the file.
THAT_RECORD_DOES_NOT_EXIST	The file Object references a file that doesn't exist.
MUST_IDENTIFY_A_FILE	The path specifies a folder and not a file.
CANNOT_RESUBMIT_SUBMITTED_ASYNC_SEARCH_TASK	The search task was already submitted and completed successfully.
ASYNC_SEARCH_DEPENDENCY_MR_ALREADY_SUBMITTED	A dependent map/reduce script is already submitted and is not complete.
ASYNC_SEARCH_DEPENDENCY_MR_INCORRECT_STATUS	The status of the deployment record for the specified dependent map/

Error Code	Thrown If
	reduce script has a value other than "Not Scheduled".
ASYNC_SEARCH_DEPENDENCY_SS_ALREADY_SUBMITTED	A dependent scheduled script is already submitted and is not complete.
ASYNC_SEARCH_DEPENDENCY_SS_INCORRECT_STATUS	The status of the deployment record for the specified dependent scheduled script has a value other than "Not Scheduled".
ASYNC_SEARCH_DEPLOYMENT_FOR_DEPENDENCY	A deployment record for the specified dependent script is not available for one of the following reasons: <ul style="list-style-type: none"> ■ A deployment record was not specified when the dependent script was created, and automatic lookup for an available deployment record failed. ■ The deployment record specified when the dependent script was created is not found.
ASYNC_SEARCH_MULTIPLE_DEPENDENCIES	The same dependent script is passed to this method more than once.
ASYNC_SEARCH_SCRIPT_ID_NOT_FOUND	The specified dependent script is not found.
ASYNC_SEARCH_SEARCH_ID_NOT_FOUND	The search task with the specified search ID is not found.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var searchTriggerTask = searchTask.submit();
...
//Add additional code
```

SearchTask.fileId

Property Description	ID of the CSV file to export search results into. <div> Note: Either this property or the SearchTask.filePath property must be set. If both are set, an error occurs. </div>
Type	The CSV file ID as a number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .

Module	N/task Module
Since	2017.1

Errors

Error Code	Thrown If
UNSUPPORTED_COMBINATION_OF_PARAMETERS	Both this property and the SearchTask.filePath property are set at the same time.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
searchTask.fileId = 18;
...
//Add additional code
```

SearchTask.filePath

Property Description	Path of the CSV file to export search results into. <div> Note: Either this property or the SearchTask.fileId property must be set. If both are set, an error occurs.</div>
Type	The CSV file path as a string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Errors

Error Code	Thrown If
UNSUPPORTED_COMBINATION_OF_PARAMETERS	Both this property and the SearchTask.fileId property are set at the same time.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
```

```
...
searchTask.filePath= 'ExportFolder/export.csv'
...
//Add additional code
```

SearchTask.inboundDependencies

Property Description	<p>Object with key/value pairs that contain information about the dependent scripts added to the search task. Use this property to verify the properties of dependent scripts after you add the scripts using SearchTask.addInboundDependency().</p> <p>This property uses nested objects to store information about each dependent script. A nested object is included for each dependent script added to the search task. The nested object contains information such as the task type, script ID, and deployment ID. It also includes the index of the script (starting at 0). Dependent scripts are indexed in the order they are added to the search task.</p> <p>For example, consider a situation in which you add a scheduled script task and a map/reduce script task to a search task as dependent scripts. After you add the dependent scripts, but before you submit the search task using SearchTask.submit(), the value of the <code>SearchTask.inboundDependencies</code> property is similar to the following:</p>
	<pre>{ "0": { "type": "task.ScheduledScriptTask", "scriptId": "customscript_as_ftr_ss", "deploymentId": "customdeploy_ss_dpl", "params": { "custscript_ss_as_srch_res": "SuiteScripts/ExportFile.csv" } }, "1": { "type": "task.MapReduceScriptTask", "scriptId": "customscript_as_ftr_mr", "deploymentId": "customdeploy_mr_dpl", "params": { "custscript_mr_as_srch_res": "SuiteScripts/ExportFile.csv" } } }</pre>
	<p>After you submit the search task, the internal IDs of the dependent scripts are added to the <code>SearchTask.inboundDependencies</code> property:</p>
	<pre>{ "0": { "type": "task.ScheduledScriptTask", "id": "SCHEDULESCRIPT_0168697b126d1705061d0d690a787755500b046a1912686b10_349d94266564827c739a2ba0a5b9d476f4097217", "scriptId": "customscript_as_ftr_ss", "deploymentId": "customdeploy_ss_dpl", "params": { "custscript_ss_as_srch_res": "SuiteScripts/ExportFile.csv" } }, "1": { "type": "task.MapReduceScriptTask", "id": "MAPREDUCETASK_0268697b126d1705061d0d69027f7b39560f01001c_7a02acb4bdebf0103120b09302170720aa57bca4", "scriptId": "customscript_as_ftr_mr", "deploymentId": "customdeploy_mr_dpl", "params": { "custscript_mr_as_srch_res": "SuiteScripts/ExportFile.csv" } } }</pre>
Type	read-only Object[]
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module

Since	2018.2
-------	--------

Errors

Error Code	Thrown If
READ_ONLY_PROPERTY	Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
// Add additional code
...
var scheduledScript = task.create({
    taskType: task.TaskType.SCHEDULED_SCRIPT
});
// Set the properties of the scheduled script task
scheduledScript.scriptId = 'customscript_as_ftr_ss';
...

var mapReduceScript = task.create({
    taskType: task.TaskType.MAP_REDUCE
});
// Set the properties of the map/reduce script task
mapReduceScript.scriptId = 'customscript_as_ftr_mr';
...

asyncTask.addInboundDependency(scheduledScript);
asyncTask.addInboundDependency(mapReduceScript);

var asyncTaskId = asyncTask.submit();

// Iterate over the dependent scripts
var p = asyncTask.inboundDependencies;
for (var key in p) {
    log.debug(key + ' > ' + p[key]);
}
...
// Add additional code
```

SearchTask.savedSearchId

Property Description	ID of the saved search to be executed during the task.
Type	The saved search ID as a number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
searchTask.savedSearchId = 51;
...
//Add additional code
```

task.SearchTaskStatus

Object Description	Encapsulates the status of an asynchronous search task (task.SearchTask) placed into the NetSuite task queue. To initiate the task and retrieve the task id, use SearchTask.submit() .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var searchTaskStatus = task.checkStatus({
    searchTaskId:51
});
if (searchTaskStatus.status === task.TaskStatus.FAILED)
...
//Add additional code
```

SearchTaskStatus.fileId

Property Description	ID of CSV file into which search results are exported.
Type	CSV file id as a number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var status = task.checkStatus({
    searchTaskId: 81
});
log.audit({
    title: 'File ID',
    details: status.fileId
});
...
//Add additional code
```

SearchTaskStatus.savedSearchId

Property Description	The ID of the saved search executed during the task.
Type	The search ID as a number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var status = task.checkStatus({
    searchTaskId: 81
});
log.audit({
    title: 'Saved Search ID',
    details: status.savedSearchId
});
...
//Add additional code
```

SearchTaskStatus.status

Property Description	Status for an asynchronous search placed in the NetSuite task queue by SearchTask.submit() . Returns a task.TaskStatus enum value.
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
log.audit({
  title: 'Status',
  details: summary.status
});

...
//Add additional code
```

SearchTaskStatus.taskId

Property Description	ID of the task.SearchTask Object. Use SearchTask.submit() to return this ID.
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2017.1

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var searchTaskId = searchTask.submit();
...
//Add additional code
```

task.WorkflowTriggerTask

Object Description	<p>Encapsulates all the properties required to asynchronously initiate a workflow. Use the <code>WorkflowTriggerTask</code> Object to create a task that initiates an instance of the specified workflow.</p> <p>The task is placed in the scheduling queue, and the workflow instance is initiated after the task reaches the top of the queue.</p> <p>To use the <code>WorkflowTriggerTask</code> Object:</p> <ul style="list-style-type: none"> Use <code>task.create(options)</code> to create the <code>WorkflowTriggerTask</code> Object. Use <code>WorkflowTriggerTask.recordType</code> to set the record type of the workflow base record. Use <code>WorkflowTriggerTask.recordId</code> to set the internal ID of the base record for the workflow. Use <code>WorkflowTriggerTask.workflowId</code> to set the internal ID of the workflow that you want to run on the record specified by the <code>recordId</code>. Optionally, use <code>WorkflowTriggerTask.params</code> to specify default values for workflow fields. Use <code>WorkflowTriggerTask.submit()</code> to submit the asynchronous workflow initiation task to the NetSuite task queue. Use the properties for the <code>WorkflowTriggerTaskStatus.status</code> object to get the status of the workflow execution. <p>Use the following guidelines with the <code>WorkflowTriggerTask</code> Object:</p> <ul style="list-style-type: none"> <code>WorkflowTriggerTask.submit()</code> does not successfully place a workflow task in the scheduling queue if an identical instance of that workflow, with the same <code>recordType</code>, <code>recordId</code>, and <code>workflowId</code>, is currently executing or already in the scheduling queue. <p>For a complete list of this object's methods and properties, see WorkflowTriggerTask Object Members.</p>
Supported Script Types	<p>Server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var workflowTask = task.create({taskType: task.TaskType.WORKFLOW_TRIGGER});
```

```

workflowTask.recordType = 'customer';
workflowTask.recordId = 107;
workflowTask.workflowId = 3;
var taskId = workflowTask.submit();
...
//Add additional code

```

WorkflowTriggerTask.submit()

Method Description	Directs NetSuite to place the asynchronous workflow initiation task into the NetSuite scheduling queue and returns a unique ID for the task. Use WorkflowTriggerTaskStatus.status to view the status of a submitted task.
Returns	the task id as a string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	20 units
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
FAILED_TO_SUBMIT_JOB_REQUEST_1	Failed to submit job request: {reason}	Task cannot be submitted.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var workflowTriggerTask = workflowTask.submit();
...
//Add additional code

```

WorkflowTriggerTask.recordType

Property Description	Record type of the workflow definition base record. For example, customer, salesorder, or lead. In the Workflow Manager, this is the record type that is specified in the Record Type field.
Type	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
workflowTask.recordType = 'customer';
...
//Add additional code
```

WorkflowTriggerTask.recordId

Property Description	Internal ID of the base record. For example, 55 or 124.
Type	number
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
workflowTask.recordId = 107;
...
//Add additional code
```

WorkflowTriggerTask.workflowId

Property Description	Internal ID (as a number), or script ID (as a string), for the workflow definition. This is the ID that appears in the ID field on the Workflow Definition Page .
Type	number string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
```

```
...
workflowTask.workflowId = 3;
...
//Add additional code
```

WorkflowTriggerTask.params

Property Description	Object that contains key/value pairs to set default values on fields specific to the workflow. These can include fields on the Workflow Definition Page or workflow and state Workflow Custom Fields .
Type	Object
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
task.params = {context: portlet}
...
//Add additional code
```

task.WorkflowTriggerTaskStatus

Object Description	Encapsulates the status of an asynchronous workflow initiation task placed into the NetSuite task queue by WorkflowTriggerTask.submit() . Use task.checkStatus(options) with the unique ID for the asynchronous workflow initiation task to get the WorkflowTriggerTaskStatus object. For a complete list of this object's properties, see WorkflowTriggerTaskStatus Object Members .
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
```



```

var workflowTaskStatus = task.checkStatus(taskId);
if (workflowTaskStatus.status === task.TaskStatus.FAILED)
...
//Add additional code

```

WorkflowTriggerTaskStatus.status

Property Description	Status for an asynchronous workflow placed in the NetSuite task queue by WorkflowTriggerTask.submit() . Returns a task.TaskStatus enum value.
Type	task.TaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Errors

Error Code	Message	Thrown If
READ_ONLY		Setting the property is attempted

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var summary = task.checkStatus(scriptTaskId);
log.audit('Status', summary.status);
...
//Add additional code

```

task.create(options)

Method Description	Creates an object for a specific task type and returns the task object. Use with the N/task Module to create a task to schedule scripts, run map/reduce scripts, import CSV files, merge duplicate records, initiate asynchronous searches, or execute asynchronous workflows.
Returns	task.ScheduledScriptTask task.MapReduceScriptTask task.CsvImportTask task.EntityDeduplicationTask task.WorkflowTriggerTask task.SearchTask
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/task Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.taskType</code>	task.TaskType	Required	The type of task object to create. Use the task.TaskType enum to set the value.	2015.2
<code>options.scriptId</code>	number string	Optional	The internal ID (as a number) or script ID (as a string) for the script record. This parameter sets the value for the ScheduledScriptTask.scriptId or MapReduceScriptTask.scriptId property. Only applicable when <code>taskType</code> is set to SCHEDULED_SCRIPT or MAP_REDUCE .	2016.2
<code>options.deploymentId</code>	number string	Optional	The internal ID (as a number) or script ID (as a string) of the script deployment record. This parameter sets the value for the ScheduledScriptTask.deploymentId or MapReduceScriptTask.deploymentId property. Only applicable when <code>taskType</code> is set to SCHEDULED_SCRIPT or MAP_REDUCE .	2016.2
<code>options.params</code>	Object	Optional	An object that represents key/value pairs that override static script parameter field values on the script deployment record. Use these parameters for the task object to programmatically pass values to the script deployment. For more information about script parameters, see the help topic Creating Script Parameters Overview . For Workflow tasks, keys can include fields on the Workflow Definition Page or workflow and state Workflow Custom Fields . This parameter sets the value for the ScheduledScriptTask.params , MapReduceScriptTask.params or WorkflowTriggerTask.params property. Only applicable when <code>taskType</code> is set to SCHEDULED_SCRIPT , MAP_REDUCE or WORKFLOW_TRIGGER .	2016.2
<code>options.importFile</code>	file.File string	Optional	A CSV file to import. Use a file.File object or a string that represents the CSV text to be imported. This parameter sets the value for the CsvImportTask.importFile property. Only applicable when <code>taskType</code> is set to CSV_IMPORT .	2016.2
<code>options.mappingId</code>	number string	Optional	The internal ID (as a number) or script ID (as a string) of a saved import map that you created when you ran the Import Assistant. See task.CsvImportTask . This parameter sets the value for the CsvImportTask.mappingId property. Only applicable when <code>taskType</code> is set to CSV_IMPORT .	2016.2

Parameter	Type	Required / Optional	Description	Since
<code>options.queueId</code>	number	Optional	<p>Overrides the Queue Number property under Advanced Options on the Import Options page of the Import Assistant. Use this property to programmatically select an import queue and improve performance during the import.</p> <div>  Note: This property is only available if you have a SuiteCloud Plus license. For more information about using multiple queues when importing CSV files, see the help topics Queue Number and Use Multiple Threads and Multiple Queues to Run CSV Import Jobs. </div> <p>This parameter sets the value for the <code>CsvImportTask.queueId</code> property. Only applicable when <code>taskType</code> is set to CSV_IMPORT.</p>	2016.2
<code>options.name</code>	string	Optional	<p>The name for the CSV import task. You can optionally set a different name for a scripted import task. In the UI, this name appears on the CSV Import Job Status page. This parameter sets the value for the <code>CsvImportTask.name</code> property. Only applicable when <code>taskType</code> is set to CSV_IMPORT.</p>	2016.2
<code>options.linkedFiles</code>	Object	Optional	<p>A map of key/value pairs that sets the data to be imported in a linked file for a multi-file import job, by referencing a file in the file cabinet or the raw CSV data to import. The key is the internal ID of the record sublist for which data is being imported and the value is either a file.File object or the raw CSV data to import. You can assign multiple types of values to the <code>linkedFiles</code> property. This parameter sets the value for the <code>CsvImportTask.linkedFiles</code> property. Only applicable when <code>taskType</code> is set to CSV_IMPORT.</p>	2016.2
<code>options.entityType</code>	string	Optional	<p>Sets the type of entity on which you want to merge duplicate records. This parameter sets the value for the <code>EntityDeduplicationTask.entityType</code> property. Only applicable when <code>taskType</code> is set to ENTITY_DEDUPLICATION. Use the <code>task.DedupeEntityType</code> enum to set the value.</p>	2016.2

Parameter	Type	Required / Optional	Description	Since
			 Note: If you set entityType to CUSTOMER, the system will automatically include prospects and leads in the task request.	
options.masterRecordId	number	Optional	<p>When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record.</p> <p>Use this property to set the ID of the master record that you want to use as the master record in the merge.</p> <div>  Important: You must also select SELECT_BY_ID for the <code>EntityDeduplicationTask.masterSelectionMode</code> property, or NetSuite ignores this setting. </div> <p>This parameter sets the value for the <code>EntityDeduplicationTask.masterRecordId</code> property.</p> <p>Only applicable when <code>taskType</code> is set to ENTITY_DEDUPLICATION.</p>	2016.2
options.masterSelectionMode	string	Optional	<p>When you merge duplicate records, you can delete all duplicates for a record or merge information from the duplicate records into the master record.</p> <p>Set this property to determine which of the duplicate records to keep or select the master record to use by ID.</p> <p>This parameter sets the value for the <code>EntityDeduplicationTask.masterSelectionMode</code> property.</p> <p>Only applicable when <code>taskType</code> is set to ENTITY_DEDUPLICATION.</p> <p>Use the <code>task.MasterSelectionMode</code> enum to set the value.</p>	2016.2
options.dedupeMode	string	Optional	<p>Sets the mode in which to merge or delete duplicate records.</p> <p>This parameter sets the value for the <code>EntityDeduplicationTask.dedupeMode</code> property.</p> <p>Only applicable when <code>taskType</code> is set to ENTITY_DEDUPLICATION.</p> <p>Use the <code>task.DedupeMode</code> enum to set the value.</p>	2016.2
options.recordIds	number[]	Optional	<p>The number array of record internal IDs to perform the merge or delete operation on.</p> <p>You can use the <code>search.duplicates(options)</code> method to identify duplicate records or create an array with record internal IDs.</p> <p>This parameter sets the value for the <code>EntityDeduplicationTask.recordIds</code> property.</p> <p>Only applicable when <code>taskType</code> is set to ENTITY_DEDUPLICATION.</p>	2016.2

Parameter	Type	Required / Optional	Description	Since
<code>options.recordType</code>	string	Optional	The record type of the workflow definition base record, such as customer, salesorder, or lead. In the Workflow Manager, this is the record type that is specified in the Record Type field. This parameter sets the value for the WorkflowTriggerTask.recordType property. Only applicable when <code>taskType</code> is set to WORKFLOW_TRIGGER .	2016.2
<code>options.recordId</code>	number	Optional	The internal ID of the base record. This parameter sets the value for the WorkflowTriggerTask.recordId property. Only applicable when <code>taskType</code> is set to WORKFLOW_TRIGGER .	2016.2
<code>options.workflowId</code>	number string	Optional	The internal ID (as a number) or script ID (as a string) for the workflow definition. This is the ID that appears in the ID field on the Workflow Definition Page . This parameter sets the value for the WorkflowTriggerTask.workflowId property. Only applicable when <code>taskType</code> is set to WORKFLOW_TRIGGER .	2016.2
<code>options.savedSearchID</code>	number	Optional	The ID of the saved search to be executed during the task.	2017.1
<code>options.fileId</code>	string	Optional	The ID of the CSV file to export search results to. See N/file Module . Note: If <code>fileId</code> is provided then the <code>filePath</code> parameter is ignored. There is no synchronization between <code>fileId</code> and <code>filePath</code> values.	2017.1
<code>options.filePath</code>	number	Optional	Path of the CSV file to export search results to. See N/file Module . Note: If <code>fileId</code> is provided then the <code>filePath</code> parameter is ignored. There is no synchronization between <code>fileId</code> and <code>filePath</code> values.	2017.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var mrTask = task.create({
  taskType: task.TaskType.MAP_REDUCE,
  scriptId: 34,
  deploymentId: 1,
```

```

    params: {doSomething: true}
  });
  ...
  //Add additional code

```

task.checkStatus(options)

Method Description	Returns a task status object associated with a specific task ID.
Returns	task.ScheduledScriptTaskStatus task.MapReduceScriptTaskStatus task.CsvImportTaskStatus task.EntityDeduplicationTaskStatus task.SearchTaskStatus task.WorkflowTriggerTaskStatus
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/task Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.taskId	task.ScheduledScriptTask task.MapReduceScriptTask task.CsvImportTask task.EntityDeduplicationTask task.SearchTask task.WorkflowTriggerTask	Required	Unique ID for the task that was generated by task.create(options) .	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```

//Add additional code
...
var taskStatus = task.checkStatus(mrTaskId);
...
//Add additional code

```

task.TaskType

Enum Description	Enumeration that holds the string values for the types of task objects supported by the N/task Module , that you can create with task.create(options) .
-------------------------	---

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- SCHEDULED_SCRIPT
- MAP_REDUCE
- CSV_IMPORT
- ENTITY_DEDUPLICATION
- SEARCH
- WORKFLOW_TRIGGER

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
var mrTask = task.create({
  taskType: task.TaskType.MAP_REDUCE
});
...
//Add additional code
```

task.TaskStatus

Enum Description	<p>Enumeration that holds the string values for the possible status of tasks created and submitted with the N/task Module.</p> <p>The following properties hold a value for <code>task.taskStatus</code>:</p> <ul style="list-style-type: none"> ■ ScheduledScriptTaskStatus.status ■ MapReduceScriptTaskStatus.status ■ CsvImportTaskStatus.status ■ EntityDeduplicationTaskStatus.status ■ SearchTaskStatus.status ■ WorkflowTriggerTaskStatus.status
-------------------------	---

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- PENDING
- PROCESSING
- COMPLETE
- FAILED

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
if (status == task.TaskStatus.COMPLETE || status == task.TaskStatus.FAILED)
...
//Add additional code
```

task.MasterSelectionMode

Enum Description	<p>Enumeration that holds the string values for supported master selection modes when merging duplicate records with task.EntityDeduplicationTask. Use this enum for the EntityDeduplicationTask.masterSelectionMode property. For more information about these values, see the help topic Merging or Deleting Duplicate Records.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- CREATED_EARLIEST

- MOST_RECENT_ACTIVITY
- MOST_POPULATED_FIELDS
- SELECT_BY_ID

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.masterSelectionMode = task.MasterSelectionMode.MOST_RECENT_ACTIVITY;
...
//Add additional code
```

task.DedupeMode

Enum Description	Enumeration that holds the string values for the available deduplication modes when merging duplicate records with <code>task.EntityDeduplicationTask</code> . Use this enum for the <code>EntityDeduplicationTask.dedupeMode</code> property. <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- MERGE
- DELETE
- MAKE_MASTER_PARENT
- MARK_AS_NOT_DUPES

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.dedupeMode = task.DedupeMode.MERGE;
```

```
...
//Add additional code
```

task.DedupeEntityType

Enum Description	<p>Enumeration that holds the string values for entity types for which you can merge duplicate records with <code>task.EntityDeduplicationTask</code>. Use this enum for the <code>EntityDeduplicationTask.entityType</code>.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- CUSTOMER
- CONTACT
- VENDOR
- PARTNER
- LEAD
- PROSPECT

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
dedupeTask.entityType = task.DedupeEntityType.CUSTOMER;
...
//Add additional code
```

task.MapReduceStage

Enum Description	<p>Enumeration that holds the string values for possible stages in <code>task.MapReduceScriptTask</code> for a map/reduce script. This enum is returned by <code>MapReduceScriptTaskStatus.stage</code>. For general information about map/reduce stages, see Map/Reduce Key Concepts and SuiteScript 2.0 Map/Reduce Script Stages.</p>
-------------------------	---

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/task Module
Since	2015.2

Values

- GET_INPUT
- MAP
- SHUFFLE
- REDUCE
- SUMMARIZE

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/task Module Script Samples](#).

```
//Add additional code
...
if (summary.stage === task.MapReduceStage.SUMMARIZE)
...
//Add additional code
```



Note: For general information about map/reduce scripts, see [SuiteScript 2.0 Map/Reduce Script Type](#).

N/transaction Module

Load the transaction module to void transactions.

When you void a transaction, the total and all the line items for the transaction are set to zero. The transaction is not removed from the system. NetSuite supports two types of voids: direct voids and voids by reversing journal. For additional information, see the help topic [Voiding, Deleting, or Closing Transactions](#).

The type of void performed with your script depends on the targeted account's preference settings:

- If the Using Reversing Journals preference is **disabled**, a **direct void** is performed.
- If the Using Reversing Journals preference is **enabled**, a **void by reversing journal** is performed.



Important: After you successfully void a transaction, you can no longer make changes to the transaction that impact the general ledger.

- [N/transaction Module Members](#)

N/transaction Module Script Sample

N/transaction Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	transaction.void(options)	number	Client and server-side scripts	Voids a transaction record.
	transaction.void.promise(options)	number	Client scripts	Voids a transaction record asynchronously.
Enum	transaction.Type	enum	Client and server-side scripts	Enumeration that holds the string values for supported record types.

N/transaction Module Script Sample

Note: This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The following examples voids a transaction.

Warning: The following sample works only in OneWorld accounts.

```
/**
 * @NApiVersion 2.x
 */
// This example shows how to void a transaction
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
    });
    // memo should be 'VOID'
    var memo = salesOrder.getValue({
        fieldId: 'memo'
    });
}
voidSalesOrder();
});
```

 **Warning:** This script sample includes hard-coded values for the purpose of illustration. To run this sample in the SuiteScript debugger, you must replace these hard-coded values with values from records in your account. For information about debugging, see the help topic [Using the SuiteScript Debugger](#).

transaction.void(options)

Method Description	<p>Method used to void a transaction record object and return an id that indicates the type of void performed.</p> <p>The type of void performed depends on the targeted account's preference settings.</p> <div> Important: After you void a transaction, you cannot make changes to the transaction that impact the general ledger.</div>
Returns	<p>An ID returned as a number.</p> <ul style="list-style-type: none">■ If a direct void is performed, returns the ID of the record voided.■ If a void by reversing journal is performed, returns the ID of the newly created voiding journal.
Supported Script Types	<p>All client and server-side scripts</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Governance	<p>10 units</p>
Module	<p>N/transaction Module</p>
Since	<p>2015.2</p>

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	number string	required	Internal ID of the specific transaction record instance to void.	2015.2
options.type	string	required	Internal ID of the type of transaction record to void	2015.2

Errors

Error Code	Message	Thrown If
INVALID_RECORD_TYPE		The type argument passed is not valid or the record type is not voidable.
THAT_RECORD_DOES_NOT_EXIST		The id argument passed is not valid.
SSS_MISSING_REQD_ARGUMENT		The type or id argument is missing.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/transaction Module Script Sample](#).

```
//Add additional code
...
var voidSalesOrderId = transaction.void({
    type: transaction.Type.SALES_ORDER,
    id: salesOrderId
});
...
//Add additional code
```

transaction.void.promise(options)

Method Description	<p>Method used to void a transaction record object asynchronously and return an id that indicates the type of void performed. The type of void performed depends on the targeted account's preference settings.</p> <div>  Important: After you void a transaction, you cannot make changes to the transaction that impact the general ledger. </div> <div>  Note: For information about the parameters and errors thrown for this method, see transaction.void(options). For additional information on promises, see Promise Object. </div>
Returns	An id returned as a number.

	<ul style="list-style-type: none"> ■ If a direct void is performed, returns the ID of the record voided. ■ If a void by reversing journal is performed, returns the ID of the newly created voiding journal.
Synchronous Version	<code>transaction.void(options)</code>
Supported Script Types	All client-side scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	10 units
Module	N/transaction Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete promise script example, see [Promise Object](#).

```
//Add additional code
...
var voidSalesOrderId = transaction.void.promise({
    type: record.Type.SALES_ORDER,
    id: salesOrderId
});
...
//Add additional code
```

transaction.Type

Enum Description	<p>Enumeration that holds the string values for supported transaction record types.</p> <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	All client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/transaction Module
Since	2015.2

Values

Transaction Record	Supported Void Type
ADVANCED_INTERCOMPANY_JOURNAL_ENTRY	Direct Void
CASH_REFUND	Direct Void
CASH_SALE	Direct Void
CHECK	Void by Reversing Journal

Transaction Record	Supported Void Type
CREDIT_MEMO	Direct Void
CUSTOMER_DEPOSIT	Direct Void
CUSTOMER_PAYMENT	Direct Void
CUSTOMER_REFUND	Direct Void and Void by Reversing Journal
ESTIMATE_QUOTE	Direct Void
EXPENSE_REPORT	Direct Void
INTERCOMPANY_JOURNAL_ENTRY	Direct Void
INVOICE	Direct Void
JOURNAL_ENTRY	Direct Void
PAYCHECK_JOURNAL	Direct Void
RETURN_AUTHORIZATION	Direct Void
SALES_ORDER	Direct Void
TRANSFER_ORDER	Direct Void
VENDOR_BILL	Direct Void
VENDOR_CREDIT	Direct Void
VENDOR_PAYMENT	Direct Void and Void by Reversing Journal
VENDOR_RETURN_AUTHORIZATION	Direct Void
WORK_ORDER	Direct Void

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/record Module Script Samples](#).

```
//Add additional code
...
var voidSalesOrderId = transaction.void({
    type: transaction.Type.SALES_ORDER,
    id: salesOrderId
});
...
//Add additional code
```

N/ui/dialog Module

Load the dialog module to create a modal dialog that persists until a button on the dialog is pressed.



Important: SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). The NetSuite UI should only be accessed using SuiteScript APIs .

■ [N/ui/dialog Module Members](#)

N/ui/dialog Module Script Samples

N/ui/dialog Module Members

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
Method	<code>dialog.alert(options)</code>	Promise	Client scripts	Creates an Alert dialog with an OK button.
	<code>dialog.confirm(options)</code>	Promise	Client scripts	Creates a Confirm dialog with OK and Cancel buttons.
	<code>dialog.create(options)</code>	Promise	Client scripts	Creates a dialog with specified buttons.

N/ui/dialog Module Script Samples

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).



Note: To debug client scripts like the following, we recommend that you 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.

The following example shows how to create an Alert dialog:

```
/**
 * @NApiVersion 2.x
 */
require(['N/ui/dialog'],
function(dialog) {
    var options = {
        title: "I am an Alert",
        message: "Press OK"
    };
    function success(result) {
        console.log("Success with value " + result);
    }
    function failure(reason) {
        console.log("Failure: " + reason);
    }

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

The following sample shows how to create a Confirmation dialog:

```
/**
 * @NApiVersion 2.x
 */
require(['N/ui/dialog'],
function(dialog) {
    var 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);
});

```

The following sample shows how to create a dialog with buttons:

```

/**
 *@NApiVersion 2.x
 */
require(['N/ui/dialog'],
function(dialog) {
    var button1 = {
        label: 'I am A',
        value: 1
    };
    var button2 = {
        label: 'I am B',
        value: 2
    };
    var button3 = {
        label: 'I am C',
        value: 3
    };
    var 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);
});

```

dialog.alert(options)

Method Description	Creates an Alert dialog with an OK button.
Returns	Promise Object . To run a callback function when the OK button is clicked, pass a function to the then portion of the Promise object. When the OK button is clicked, true is passed to

	the callback. You do not have to utilize the Promise object unless there is an action you want performed after the user clicks the OK button.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/dialog Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	optional	The alert dialog title. This value defaults to an empty string.	2016.1
options.message	string	optional	The content of the alert dialog. This value defaults to an empty string.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/dialog Module Script Samples](#).

```
//Add additional code
...
function success(result) { console.log('Success with value: ' + result) }
function failure(reason) { console.log('Failure: ' + reason) }

dialog.alert({
  title: 'Alert',
  message: 'Click OK to continue.'
}).then(success).catch(failure);
...
//Add additional code
```

dialog.confirm(options)

Method Description	Creates a Confirm dialog with OK and Cancel buttons.
Returns	Promise Object . To run a callback function when the OK button is pressed, pass a function to the then portion of the Promise object. The value of the pressed button, where OK is true and Cancel is false , is passed to the callback.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/dialog Module

Since	2016.1
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	optional	The confirmation dialog title. This value defaults to an empty string.	2016.1
options.message	string	optional	The content of the confirmation dialog. This value defaults to an empty string.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/dialog Module Script Samples](#).

```
//Add additional code
...
var 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);

...
//Add additional code
```

dialog.create(options)

Method Description	Creates a dialog with specified buttons.
Returns	Promise Object . To run a callback function when a button is pressed, pass a function to the then portion of the Promise object. The value of the button pressed is passed to the callback.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/dialog Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.buttons	string[]	optional	A list of buttons to include in the dialog. Each item in the button list must be a Javascript Object that contains a label and a value property. By default, a single button with the label OK and the value true is used.	2016.1
options.title	string	optional	The dialog title. This value defaults to an empty string.	2016.1
options.message	string	optional	The content of the dialog. This value defaults to an empty string.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/dialog Module Script Samples](#).

```
//Add additional code
...
var options = {
  title: 'Dialog',
  message: 'Click a button to continue.',
  buttons: [
    { label: '1', value: 1 },
    { label: '2', value: 2 },
    { label: '3', value: 3 }
  ]
};

function success(result) { console.log('Success with value: ' + result) }
function failure(reason) { console.log('Failure: ' + reason) }

dialog.create(options).then(success).catch(failure);
...
//Add additional code
```

N/ui/message module

Load the message module to display a message at the top of the screen under the menu bar.

 **Important:** SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). The NetSuite UI should only be accessed using SuiteScript APIs .

- [N/ui/message Members](#)
- [Message Object Members](#)
- [N/ui/message Module Script Sample](#)

N/ui/message Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	message.Message	void	Client scripts	Encapsulates the Message object that gets created when calling the create method.
Method	message.create(options)	Message	Client scripts	Creates a message that can be displayed or hidden near the top of the page.
Enum	message.Type	enum	Client scripts	Indicates the type of message to display, which specifies the background color of the message and other message indicators.

Message Object Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Message.hide()	void	Client scripts	Hides the message.
	Message.show()	void	Client scripts	Shows the message.

N/ui/message Module Script Sample

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

To debug client scripts like the following, we recommend that you 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.

The following example shows how to create a confirmation message:

```
/**
 *@NApiVersion 2.x
 */
require(['N/ui/message'],
function(message) {
    var myMsg = message.create({
        title: "My Title",
        message: "My Message",
        type: message.Type.CONFIRMATION
    });

    // will disappear after 5s
    myMsg.show({
        duration: 5000
    });

    var 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

var myMsg3 = message.create({
  title: "My Title 3",
  message: "My Message 3",
  type: message.Type.WARNING
});

myMsg3.show(); // will stay up until hide is called.
});
```

message.Message

Object Description	Encapsulates the Message object that gets created when calling the create method. For a complete list of this object's methods, see Message Object Members .
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/ui/message module
Since	2016.1

Message.hide()

Method Description	Hides the message.
Returns	void
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/message module
Since	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/message Module Script Sample](#).

```
//Add additional code
...
var myMsg = message.create({
  title: "My Title 2",
  message: "My Message 2",
  type: message.Type.INFORMATION
});
myMsg.show();
setTimeout(myMsg.hide(), 15000); // hide the message after 15s
...
//Add additional code
```

Message.show()

Method Description	Shows the message.
Returns	void
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/message module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.duration	int	optional	The amount of time, in milliseconds, to show the message. The default is 0, which shows the message until Message.hide() is called. If you specify a duration for message.create() and message.show(), the value from the message.show() method call takes precedence.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/message Module Script Sample](#).

```
//Add additional code
...
var myMsg = message.create({
  title: "My Title 2",
  message: "My Message 2",
  type: message.Type.INFORMATION
});
myMsg.show({ duration : 1500 });
...
//Add additional code
```

message.create(options)

Method Description	Creates a message that can be displayed or hidden near the top of the page.
Returns	message.Message .
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Governance	None
Module	N/ui/message module

Since	2016.1
-------	--------

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.type	message.Type	required	The message type.	2016.1
options.title	string	optional	The message title. This value defaults to an empty string.	2016.1
options.message	string	optional	The content of the message. This value defaults to an empty string.	2016.1
options.duration	int	optional	The amount of time, in milliseconds, to show the message. The default is 0, which shows the message until <code>Message.hide()</code> is called. If you specify a duration for <code>message.create()</code> and <code>message.show()</code> , the value from the <code>message.show()</code> method call takes precedence.	2018.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/message Module Script Sample](#).

```
//Add additional code
...
var myMsg = message.create({
    title: "My Title",
    message: "My Message",
    type: message.Type.CONFIRMATION
});
...
//Add additional code
```

message.Type

Enum Description	Indicates the type of message to display, which specifies the background color of the message and other message indicators.
Supported Script Types	Client scripts For more information, see SuiteScript 2.0 Client Script Type .
Module	N/ui/message module
Since	2016.1

Values

Value	Color
CONFIRMATION	A green background with a checkmark icon.

Value	Color
INFORMATION	A blue background with an Information icon.
WARNING	A yellow background with a Warning icon.
ERROR	A red background with an X icon.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/message Module Script Sample](#).

```
//Add additional code
...
var myMsg = message.create({
    title: "My Title",
    message: "My Message",
    type: message.Type.CONFIRMATION
});
myMsg.show();
...
//Add additional code
```

N/ui/serverWidget Module

Load the serverWidget module when you want to work with the user interface within NetSuite. You can use Suitelets to build custom pages and wizards that have a NetSuite look-and-feel. You can also create various components of the NetSuite UI (for example, forms, fields, sublists, tabs).



Important: SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). The NetSuite UI should only be accessed using SuiteScript APIs .



Important: When you add a UI object to an existing NetSuite page, to minimize the occurrence of field/object name conflicts, the internal ID that references the object must be prefixed with `custpage`.

- [N/ui/serverWidget Module Members](#)
- [Assistant Object Members](#)
- [AssistantStep Object Members](#)
- [Button Object Members](#)
- [Field Object Members](#)
- [FieldGroup Object Members](#)
- [Form Object Members](#)
- [List Object Members](#)
- [ListColumn Object Members](#)
- [Sublist Object Members](#)
- [Tab Object Members](#)

■ N/ui/serverWidget Module Script Samples

N/ui/serverWidget Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	<code>serverWidget.Assistant</code>	Object	Suitelets	Encapsulates a scriptable, multi-step NetSuite assistant.
	<code>serverWidget.AssistantStep</code>	Object	Suitelets	Encapsulates a step within a custom NetSuite assistant.
	<code>serverWidget.Button</code>	Object	Suitelets	Encapsulates a button that appears in a UI object.
	<code>serverWidget.Field</code>	Object	Suitelets	Encapsulates a NetSuite field.
	<code>serverWidget.FieldGroup</code>	Object	Suitelets	Encapsulates a field group.
	<code>serverWidget.Form</code>	Object	Suitelets	Encapsulates a NetSuite form.
	<code>serverWidget.List</code>	Object	Suitelets	Encapsulates a list.
	<code>serverWidget.ListColumn</code>	Object	Suitelets	Encapsulates list columns.
	<code>serverWidget.Sublist</code>	Object	Suitelets	Encapsulates a NetSuite sublist.
	<code>serverWidget.Tab</code>	Object	Suitelets	Encapsulates NetSuite tabs and subtabs.
Method	<code>serverWidget.createAssistant(options)</code>	<code>serverWidget.Assistant</code>	Suitelets	Creates and returns a new assistant object.
	<code>serverWidget.createForm(options)</code>	<code>serverWidget.Form</code>	Suitelets	Creates and returns a new form object.
	<code>serverWidget.createList(options)</code>	<code>serverWidget.List</code>	Suitelets	Instantiates a List object (specifying the title, and whether to hide the navigation bar)
Enum	<code>serverWidget.AssistantSubmitAction</code>	string (read-only)	Suitelets	Holds the string values for submit actions performed by the user.
	<code>serverWidget.FieldBreakType</code>	string (read-only)	Suitelets	Holds the string values for supported field break types. This enum is used to set the value of the <code>Field.updateBreakType(options)</code> property.
	<code>serverWidget.FieldDisplayType</code>	string (read-only)	Suitelets	Holds the string values for supported field display types. This enum is used to set the value of the <code>Field.updateDisplayType(options)</code> property.
	<code>serverWidget.FieldLayoutType</code>	string (read-only)	Suitelets	Holds the string values for the supported types of field layouts. This enum is used to set the value of the <code>Field.updateLayoutType(options)</code> property.
	<code>serverWidget.FieldType</code>	string (read-only)	Suitelets	Holds the values for supported field types. This enum is

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				used to set the value of the Field.type property.
	serverWidget.FormPageLinkType	string (read-only)	Suitelets	Holds the string values for supported page link types on a form. This enum is used to set the value of the <code>type</code> parameter for Form.addPageLink(options) .
	serverWidget.LayoutJustification	string (read-only)	Suitelets	Holds the string values for supported justification layouts. This enum is used to set the value of the <code>align</code> parameter when List.addColumn(options) is called.
	serverWidget.ListStyle	string (read-only)	Suitelets	Holds the string values for supported list styles. This enum is used to set the value of the <code>style</code> property.
	serverWidget.SublistDisplayType	string (read-only)	Suitelets	Holds the string values for supported sublist display types. This enum is used to set the value of the <code>Sublist.displayType</code> property.
	serverWidget.SublistType	string (read-only)	Suitelets	Holds the string values for valid sublist types. This enum is used to define the <code>type</code> parameter when Form.addSublist(options) is called.

Assistant Object Members

The following members are called on the [serverWidget.Assistant](#) object.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
Method	Assistant.addField(options)	serverWidget.Field	Suitelets	Adds a field to an assistant.
	Assistant.addFieldGroup(options)	serverWidget.FieldGroup	Suitelets	Adds a field group to an assistant.
	Assistant.addStep(options)	serverWidget.AssistantStep	Suitelets	Adds a step to an assistant.
	Assistant.addSublist(options)	serverWidget.Sublist	Suitelets	Adds a sublist to an assistant.
	Assistant.getField(options)	serverWidget.Field	Suitelets	Gets a field object.
	Assistant.getFieldGroup(options)	serverWidget.FieldGroup	Suitelets	Gets a field group object.
	Assistant.getFieldGroupIds()	string	Suitelets	Gets all the field group IDs in an assistant.
	Assistant.getFieldIds()	string	Suitelets	Gets all the field IDs in an assistant.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
	Assistant.getFieldIdsByFieldGroup(fieldGroup)	string[]	Suitelets	Gets all field IDs in the assistant field group.
	Assistant.getLastAction()	string	Suitelets	Gets the last action submitted by the user.
	Assistant.getLastStep()	serverWidget.AssistantStep	Suitelets	Gets the step that the last submitted action came from.
	Assistant.getNextStep()	serverWidget.AssistantStep	Suitelets	Gets the next step prompted by the assistant.
	Assistant.getStep(options)	serverWidget.AssistantStep	Suitelets	Returns a step in an assistant.
	Assistant.getStepCount()	serverWidget.AssistantStep	Suitelets	Gets the total count of steps in the assistant.
	Assistant.getSteps()	serverWidget.AssistantStep[]	Suitelets	Gets all the steps in the assistant.
	Assistant.getSublist(options)	serverWidget.Sublist	Suitelets	Get a Sublist object from its ID.
	Assistant.getSublistIds()	string	Suitelets	Gets all the sublist IDs in an assistant.
	Assistant.hasErrorHtml()	boolean true false	Suitelets	Indicates whether the assistant threw an error.
	Assistant.isFinished()	boolean true false	Suitelets	Sets the status of the assistant. If set to true, the assistant is finished.
	Assistant.sendRedirect(options)	void	Suitelets	Manages redirects in an assistant.
	Assistant.setSplash(options)	void	Suitelets	Define a splash message.
	Assistant.updateDefaultValues(values)	void	Suitelets	Sets the default values of an array of fields that are specific to the assistant.
Property	Assistant.clientScriptFileId	number	Suitelets	The file cabinet ID of client script file to be used in this assistant.
	Assistant.clientScriptModulePath	string	Suitelets	The relative path to the client script file to be used in this assistant.
	Assistant.currentStep	serverWidget.AssistantStep	Suitelets	Identifies the current step.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
	Assistant.errorHtml	string	Suitelets	The error message text.
	Assistant.finishedHtml	string	Suitelets	The text displayed after an assistant is finished.
	Assistant.hideAddToShortcutsLink	boolean true false	Suitelets	Indicates whether the Add to Shortcuts Link is displayed in the UI.
	Assistant.hideStepNumber	boolean true false	Suitelets	Indicates whether the current and total step numbers are displayed in the UI.
	Assistant.isNotOrdered	boolean true false	Suitelets	Indicates whether assistant steps are ordered or unordered.
	Assistant.title	string	Suitelets	The title of an assistant.

AssistantStep Object Members

The following members are called on the `serverWidget.AssistantStep` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	AssistantStep.getFieldIds()	string	Suitelets	Gets all the field IDs in an assistant step.
	AssistantStep.getLineCount(options)	number	Suitelets	Gets the number of lines previously entered by a user in a step.
	AssistantStep.getLineCount(options)	string	Suitelets	Gets all the field IDs in a list.
	AssistantStep.getSubmittedSublistIds()	string	Suitelets	Gets the IDs for all the sublist fields (line items) in a step.
	AssistantStep.getSublistValue(options)	string	Suitelets	Gets the current value of a sublist field (line item) in a step.
	AssistantStep.getValue(options)	string	Suitelets	Gets the current value of a field.
Property	AssistantStep.helpText	string	Suitelets	The help text for a step.
	AssistantStep.id	string	Suitelets	The internal ID of the step.
	AssistantStep.label	string	Suitelets	The label for a step.
	AssistantStep.stepNumber	number	Suitelets	Indicates where this step appears sequentially in an assistant.

Button Object Members

The following members are called on the `serverWidget.Button` object.

Member Type	Name	Property Type	Supported Script Types	Description
Property	<code>Button.isDisabled</code>	boolean <code>true</code> <code>false</code>	Suitelets	Indicates whether a button is grayed-out and disabled.
	<code>Button.isHidden</code>	boolean <code>true</code> <code>false</code>	Suitelets	Indicates whether the button is hidden in the UI.
	<code>Button.label</code>	string	Suitelets	The label for the button.

Field Object Members

The following members are called on the `serverWidget.Field` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Field.addSelectOption(options)</code>	void	Suitelets	Adds a select option to a dropdown list for a selectable field.
	<code>Field.getSelectOptions(options)</code>	object[]	Suitelets	Returns the internal ID and label of the options for a select field as name/value pairs.
	<code>Field.setHelpText(options)</code>	void	Suitelets	Sets the help text that appears in the field help popup.
	<code>Field.updateBreakType(options)</code>	<code>serverWidget.Field</code>	Suitelets	Updates the break type used to add a break in flow layout for the field.
	<code>Field.updateDisplaySize(options)</code>	<code>serverWidget.Field</code>	Suitelets	Updates the height and width for the field.
	<code>Field.updateDisplayType(options)</code>	<code>serverWidget.Field</code>	Suitelets	Updates the type of display for the field.
	<code>Field.updateLayoutType(options)</code>	<code>serverWidget.Field</code>	Suitelets	Updates the layout type for the field.
Property	<code>Field.alias</code>	string	Suitelets	The alias used to set the field value.
	<code>Field.defaultValue</code>	string	Suitelets	The default value for the field.
	<code>Field.id</code>	string	Suitelets	The internal ID for the field.
	<code>Field.isMandatory</code>	boolean <code>true</code> <code>false</code>	Suitelets	Indicates whether the field is mandatory.
	<code>Field.label</code>	string	Suitelets	Gets or sets the label for the field.
	<code>Field.linkText</code> <code>FieldGroup.isBorderHidden</code>	string	Suitelets	The text displayed for a link in place of the URL.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Field.maxLength	number	Suitelets	The maximum length, in characters, for the field.
	Field.padding	number	Suitelets	The number of empty vertical character spaces above the field.
	Field.richTextHeight	number	Suitelets	The height of a rich text field, in pixels.
	Field.richTextWidth	number	Suitelets	The width of a rich text field, in pixels.
	Field.type	string	Suitelets	The type of field.

FieldGroup Object Members

The following members are called on the [serverWidget.FieldGroup](#) object.

Member Type	Name	Property Type	Supported Script Types	Description
Property	FieldGroup.isBorderHidden	boolean true false	Suitelets	Indicates whether a border appears around the field group.
	FieldGroup.isCollapsible	boolean true false	Suitelets	Indicates whether the field group is collapsible.
	FieldGroup.isCollapsed	boolean true false	Suitelets	Indicates whether the field group is initially collapsed or expanded in the default view.
	FieldGroup.isSingleColumn	boolean true false	Suitelets	Indicates whether the field group is aligned.
	FieldGroup.label	string	Suitelets	The label for the field group.

Form Object Members

The following members are called on the [serverWidget.Form](#) object.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
Method	Form.addButton(options)	serverWidget.Button	Suitelets	Adds a button to the form.
	Form.addCredentialField(options)	serverWidget.Field	Suitelets	Adds a field that store credentials in NetSuite for invoking services provided by third parties.
	Form.addField(options)	serverWidget.Field	Suitelets	Adds a field to the form.
	Form.addFieldGroup(options)	serverWidget.FieldGroup	Suitelets	Adds a group of fields to the form.
	Form.addPageInitMessage(options)	void	Suitelets	Shows a message on a form in view mode. You can

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
				use this method to show a message on a form based on its user event script context.
	Form.addPageLink(options)	void	Suitelets	Adds a link to a form.
	Form.addResetButton(options)	serverWidget.Button	Suitelets	Adds a reset button to a form that clears user input.
	Form.addSecretKeyField(options)	serverWidget.Field	Suitelets	Add a secret key field to the form.
	Form.addSublist(options)	serverWidget.Sublist	Suitelets	Adds a sublist to the form.
	Form.addSubmitButton(options)	serverWidget.Button	Suitelets	Adds a submit button to a form that saves user inputs.
	Form.addSubtab(options)	serverWidget.Tab	Suitelets	Adds a subtab to a form.
	Form.addTab(options)	serverWidget.Tab	Suitelets	Adds a tab to a form.
	Form.getButton(options)	serverWidget.Button	Suitelets	Returns a button by internal ID.
	Form.getField(options)	serverWidget.Field	Suitelets	Returns a field by internal ID.
	Form.getSublist(options)	serverWidget.Sublist	Suitelets	Returns a sublist by internal ID.
	Form.getSubtab(options)	serverWidget.Tab	Suitelets	Returns a subtab by internal ID.
	Form.getTab(options)	serverWidget.Tab	Suitelets	Returns a tab object from its internal ID.
	Form.getTabs()	serverWidget.Tab[]	Suitelets	Returns an array of all the tabs in a form.
	Form.insertField(options)	void	Suitelets	Inserts a field before another field within a form.
	Form.insertSublist(options)	serverWidget.Sublist	Suitelets	Inserts a sublist before another sublist on a form.
	Form.insertSubtab(options)	serverWidget.Tab	Suitelets	Inserts a subtab before another subtab on a form.
	Form.insertTab(options)	serverWidget.Tab	Suitelets	Inserts a tab before another tab on a form.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
	Form.removeButton(options)	void	Suitelets and beforeLoad user events	Removes a button from a form.
	Form.updateDefaultValues(options)	void	Suitelets	Sets the default values of many fields on a form.
Property	Form.clientScriptFileId	number	Suitelets	The file cabinet ID of client script file to be used in this form.
	Form.clientScriptModulePath	string	Suitelets	The relative path to the client script file to be used in this form.
	Form.title	string	Suitelets	The title used for the form.

List Object Members

The following members are called on the [serverWidget.List](#) object.

Member Type	Name	Property Type / Method Return Type	Supported Script Types	Description
Method	List.addButton(options)	serverWidget.Button	Suitelets	Adds a button to a list.
	List.addColumn(options)	serverWidget.ListColumn	Suitelets	Adds a column to a list.
	List.addEditColumn(options)	serverWidget.ListColumn	Suitelets	Adds a column containing Edit or Edit/View links to a Suitelet or Portlet list.
	List.addPageLink(options)	void	Suitelets	Adds a link to a list.
	List.addRow(options)	void	Suitelets	Adds a single row to a list.
	List.addRows(options)	void	Suitelets	Adds multiple rows to a list.
Property	List.clientScriptFileId	number	Suitelets	The file cabinet ID of client script file to be used in this list.
	List.clientScriptModulePath	string	Suitelets	The relative path to the client script file to be used in this list.
	List.style	string	Suitelets	Sets the display style for this list.
	List.title	string	Suitelets	Sets the List title.

ListColumn Object Members

The following members are called on the [serverWidget.ListColumn](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	ListColumn.addParamToURL(options)	serverWidget.ListColumn	Suitelets	Adds a URL parameter (optionally defined per row) to the list column's URL.
	ListColumn.setURL(options)	serverWidget.ListColumn	Suitelets	Sets the base URL for the list column.
Property	ListColumn.label	string	Suitelets	The label of this list column.

Sublist Object Members

The following members are called on the [serverWidget.Sublist](#) object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Sublist.addButton(options)	serverWidget.Button	Suitelets	Adds a button to a sublist.
	Sublist.addField(options)	serverWidget.Field	Suitelets	Add a field to a sublist.
	Sublist.addMarkAllButtons()	serverWidget.Button	Suitelets	Adds a Mark All or Unmark All button.
	Sublist.addRefreshButton()	serverWidget.Button	Suitelets	Adds a Reset button.
	Sublist.getField(options)	serverWidget.Field	Suitelets	Returns a Field object on a specified sublist.
	Sublist.getSublistValue(options)	string	Suitelets	Gets a field value on a sublist.
	Sublist.setSublistValue(options)	void	Suitelets	Sets the value of a sublist field.
	Sublist.updateTotallingFieldId(options)	serverWidget.Sublist	Suitelets	Updates the ID of a field designated as a totalling column, which is used to calculate and display a running total for the sublist.
	Sublist.updateUniqueFieldId(options)	serverWidget.Sublist	Suitelets	Updates a field ID that is to have unique values across the rows in the sublist.
Property	Sublist.displayType	string	Suitelets	The display style for a sublist.
	Sublist.helpText	string	Suitelets	The inline help text for a sublist.
	Sublist.label	string	Suitelets	The label for a sublist.
	Sublist.lineCount	number (read-only)	Suitelets	The number of line items in a sublist.

Tab Object Members

The following members are called on the [serverWidget.Tab](#) object.

Member Type	Name	Value Type	Supported Script Types	Description
Property	Tab.helpText	string	Suitelets	The inline help text for a tab or subtab.
	Tab.label	string	Suitelets	The label for a tab or subtab.

N/ui/serverWidget Module Script Samples

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

Example 1

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

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
function(serverWidget) {
    function onRequest(context) {
        if (context.request.method === 'GET') {
            var form = serverWidget.createForm({
                title: 'Simple Form'
            });
            var 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'
            });
            var select = form.addField({
                id: 'selectfield',
                type: serverWidget.FieldType.SELECT,
                label: 'Select'
            });
            select.addSelectOption({
                value: 'a',
                text: 'Albert'
            });
        }
    }
});
```

```

    });
    select.addSelectOption({
        value: 'b',
        text: 'Baron'
    });
    var 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'
    });

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

    context.response.write('You have entered: ' + textField + ' ' + dateField + ' '
        + currencyField + ' ' + selectField + ' ' + sublistField1 + ' ' + sublistField2);
}

return {
    onRequest: onRequest
};
});

```

Example 2

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

```

/**
 * @ApiVersion 2.0
 * @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 5," +
    "1 = Greatly Unsatisfied and 5 = Greatly Satisfied.</p><br><br>";

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_rdoproducrating',
        type: serverWidget.FieldType.RADIO,
        label: '4',
        source: 'p4'
    }).updateLayoutType({
        layoutType: serverWidget.FieldLayoutType.MIDROW
    });
    form.addField({
        id: 'custpage_rdoproducrating',
        type: serverWidget.FieldType.RADIO,
        label: '5',
        source: 'p5'
    }).updateLayoutType({
        layoutType: serverWidget.FieldLayoutType.ENDROW
    });
    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.ENDROW
        });

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

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

```

serverWidget.Assistant

Object Description	<p>Encapsulates a scriptable, multi-step NetSuite assistant. An assistant contains a series of step that a user must complete to accomplish a larger goal. An assistant can be sequential, or non-sequential and include optional steps. Each page of the assistant is defined by a step. All data and states for an assistant are tracked automatically throughout the user's session until completion of the assistant.</p> <p>You can create a new assistant with the serverWidget.createAssistant(options) method. After you create an <code>Assistant</code> object, you can:</p> <ul style="list-style-type: none"> ■ Build and run an assistant in your NetSuite account. ■ Add a variety of scriptable elements to the assistant including fields, steps, buttons, tabs, and sublists. <p>For a complete list of this object's methods and properties, see Assistant Object Members.</p>
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
//Add additional code
```

Assistant.addField(options)

Method Description	Adds a field to an assistant. Use fields to record or display information specific to your needs.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID for this field.	2015.2
<code>options.label</code>	string	required	The label for this field.	2015.2
<code>options.type</code>	string	required	The field type. Use the serverWidget.FieldType enum to set this value. <div> Note: If you have set the <code>type</code> parameter to <code>SELECT</code>, and you want to add custom options to the select field, you must set <code>source</code> to <code>NULL</code>. Then, when a value is specified, the value will populate the options from the source. </div> <div> Important: Long text fields created with SuiteScript have a character limit of 100,000. Long text fields created with Suitebuilder have a character limit of 1,000,000. </div>	2015.2
<code>options.source</code>	string	optional	The internalId or scriptId of the source list for this field. Use the serverWidget.FieldType enum to set this value.	2015.2

Parameter	Type	Required / Optional	Description	Since
			<div>  Note: If you want to add custom options on a select field, you must set the source parameter to NULL. </div> <div>  Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with <code>Field.addSelectOption(options)</code>. The select values are determined by the source record or list. </div>	
<code>options.container</code>	string	optional	The internal ID of the field group to place this field in.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addField({
    id : 'idname',
    type : serverWidget.FieldType.TEXT,
    label : 'Sample label'
});
...
//Add additional code
```

Assistant.addFieldGroup(options)

Method Description	Adds a field group to the assistant. A field group is a collection of fields that can be displayed in a one or two column format. Assign a field to a field group in order to label, hide or collapse a group of fields. By default, the field group is collapsible and appears expanded on the assistant page. To change this behavior, set the FieldGroup.isCollapsed and FieldGroup.isCollapsible properties.
Returns	serverWidget.FieldGroup object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID for the field group.	2015.2
options.label	string	required	The label for the field group.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addFieldGroup({
    id : 'idname',
    label : 'Sample label'
});
...
//Add additional code
```

Assistant.addStep(options)

Method Description	Adds a step to an assistant. Steps define each page of the assistant. Use Assistant.isNotOrdered to control if the steps must be completed sequentially or in no specific order. If you want to create help text for the step, you can use AssistantStep.helpText on the object returned.
Returns	serverWidget.AssistantStep object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID for this step (for example, 'entercontacts').	2015.2

Parameter	Type	Required / Optional	Description	Since
options.label	string	required	The label for this step (for example, 'Enter Contacts'). By default, the step appears vertically in the left panel of the assistant.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
...
//Add additional code
```

Assistant.addSublist(options)

Method Description	Adds a sublist to an assistant. <div> Note: Only inline editor sublists are added. Other sublist types are not supported. </div>
Returns	<code>serverWidget.Sublist</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID for the sublist.	2015.2
options.label	string	required	The label for the sublist.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.type	string	required	The type of sublist to add. Currently, only the sublist type of <code>INLINEEDITOR</code> can be added. For more information about this type of sublist, see serverWidget.SublistType .	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addSublist({
    id : 'idname',
    label : 'Sample label',
    type : serverWidget.SublistType.INLINEEDITOR
});
...
//Add additional code
```

Assistant.getField(options)

Method Description	Returns a field object on an assistant page.
Returns	serverWidget.Field
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of the field.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```

...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addField({
    id : 'idname',
    type : serverWidget.FieldType.TEXT,
    label : 'Sample label'
});
var field = assistant.getField({
    id: 'idname'
});
...
//Add additional code

```

Assistant.getFieldGroup(options)

Method Description	Returns a field group on an assistant page.
Returns	serverWidget.FieldGroup object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of the field group.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addFieldGroup({
    id : 'idname',
    label : 'Sample label'
});

```

```
var fieldgroup = assistant.getFieldGroup({
    id: 'idname'
});
...
//Add additional code
```

Assistant.getFieldGroupIds()

Method Description	Retrieves all the internal IDs for field groups in an assistant.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addFieldGroup({
    id : 'idname',
    label : 'Sample label'
});
var fieldgroupid = assistant.getFieldGroupIds();
...
//Add additional code
```

Assistant.getFieldIds()

Method Description	Gets all the internal IDs for fields in an assistant.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addField({
    id : 'idname',
    label : 'Sample label'
});
var fieldid = assistant.getFieldIds();
...
//Add additional code
```

Assistant.getFieldIdsByFieldGroup(fieldGroup)

Method Description	Gets all field IDs in the assistant field group.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
fieldGroup	string	required	The internal ID of the field group.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Sample label'
});
```



```

});
assistant.addField({
    id : 'idname',
    label : 'Sample label'
});
var fieldid = assistant.getFieldIdsByFieldGroup({
    fieldGroup : 'fieldgroupid'
});
...
//Add additional code

```

Assistant.getLastAction()

Method Description	Gets the last action taken by the user. To identify the step that the last action came from, use Assistant.getLastStep() .
Returns	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
if (assistant.getLastAction() == serverWidget.AssistantSubmitAction.CANCEL) {
    ...
}
...
//Add additional code

```

Assistant.getLastStep()

Method Description	Gets the step that the last submitted action came from.
Returns	A serverWidget.AssistantStep object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .

Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
var lastStep = assistant.getLastStep();
...
//Add additional code
```

Assistant.getNextStep()

Method Description	Gets the next step corresponding to the user's last submitted action in the assistant. If you need information about the last step, use Assistant.getLastStep() before you use this method.
Returns	serverWidget.AssistantStep object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
var nextStep = assistant.getNextStep();
...
//Add additional code
```

Assistant.getStep(options)

Method Description	Returns a step in an assistant.
Returns	serverWidget.AssistantStep object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of the step.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var firststep = assistant.getStep({
    id: 'idname'
});
...
//Add additional code
```

Assistant.getStepCount()

Method Description	Gets the total number of steps in an assistant.
Returns	The total count of assistant steps as a number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module

Since	2015.2
-------	--------

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
assistant.addStep({
  id : 'idname',
  label : 'Sample label'
});
var numSteps = assistant.getStepCount();
...
//Add additional code
```

Assistant.getSteps()

Method Description	Gets all the steps in an assistant.
Returns	serverWidget.AssistantStep[] object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
assistant.addStep({
  id : 'idname',
  label : 'Sample label'
});
var steps = assistant.getSteps();
...
//Add additional code
```

Assistant.getSublist(options)

Method Description	Returns a sublist in an assistant.
Returns	serverWidget.Sublist object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of the sublist.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addSublist({
    id : 'idname',
    label : 'Sample label',
    type : serverWidget.SublistType.LIST
});
var sublist = assistant.getSublist({
    id : 'idname'
});
...
//Add additional code
```

Assistant.getSublistIds()

Method Description	Gets the IDs for all the sublists in an assistant.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None

Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addSublist({
    id : 'idname',
    label : 'Sample label',
    type : serverWidget.SublistType.LIST
});
var sublistid = assistant.getSublistIds();
...
//Add additional code
```

Assistant.hasErrorHtml()

Method Description	Determine whether an assistant has an error message to display for the current step.
Returns	boolean <code>true</code> if <code>Assistant.errorHtml</code> contains a value.
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
if (assistant.hasErrorHtml()) {
    ...
}
```

```
...
//Add additional code
```

Assistant.isFinished()

Method Description	Indicates whether all steps in an assistant are completed. If set to <code>true</code> , the assistant is finished and a completion message displays. To set the text for the completion message, use the Assistant.finishedHtml property.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
...
if (assistant.isFinished()) {
    ...
}
...

//Add additional code
```

Assistant.sendRedirect(options)

Method Description	Manages redirects in an assistant. This method also addresses the case in which one assistant redirects to another assistant. In this scenario, the second assistant must return to the first assistant if the user Cancels or Finishes. This method, when used in the second assistant, ensures that users are redirected back to the first assistant.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.response	response	required	The response that redirects the user.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title: 'Small Business Setup Assistant',
    hideNavBar: true
});
if (request.method === 'POST') {
    if (assistant.getLastAction() === 'finish') {
        assistant.finishedHtml = 'Completed!';
        assistant.sendRedirect({
            response: response
        });
    }
}
...
//Add additional code
```

Assistant.setSplash(options)

Method Description	Defines a splash message.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	required	The title of the splash screen.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.text1	string	required	Text for the splash screen	2015.2
options.text2	string	optional	Text for a second column on the splash screen, if desired.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.setSplash({
    title: "Welcome Title!",
    text1: "An explanation of what this assistant accomplishes.",
    text2: "Some parting words."
});
...
//Add additional code
```

Assistant.updateDefaultValues(values)

Method Description	Sets the default values of an array of fields that are specific to the assistant.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
values	object[]	required	An array of fields to update.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
```

```

        title : 'Simple Assistant'
    });
    assistant.addField({
        id : 'idname',
        type : serverWidget.FieldType.TEXT,
        label : 'Sample label'
    });
    assistant.updateDefaultValues({
        idname : "New Default Value"
    });
    ...
    //Add additional code

```

Assistant.clientScriptFileId

Property Description	The file cabinet ID of client script file to be used in this assistant.
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the <code>Assistant.clientScriptModulePath</code> property value has already been specified. For more information, see Assistant.clientScriptModulePath .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.clientScriptId = 32;
...
//Add additional code

```

Assistant.clientScriptModulePath

Property Description	The relative path to the client script file to be used in this assistant.
-----------------------------	---

Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2016.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the <code>Assistant.clientScriptFileId</code> property value has already been specified. For more information, see Assistant.clientScriptFileId .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
objAssistant.clientScriptModulePath = 'SuiteScripts/assistantBehavior.js';
...
//Add additional code
```

Assistant.currentStep

Property Description	Identifies the current step. You can set any step as the current step.
Type	serverWidget.AssistantStep (read-only)
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
```

```

        title : 'Simple Assistant'
    });
    assistant.addStep({
        id : 'idname',
        label : 'Sample label'
    });
    var step = assistant.currentStep;
    ...
    //Add additional code

```

Assistant.errorHtml

Property Description	Error message text for the current step. Optionally, you can use HTML tags to format the message.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.errorHtml = "You have <b>not</b> filled out the required fields. Please go back.";
...
//Add additional code

```

Assistant.finishedHtml

Property Description	The text to display after the assistant finishes. For example “You have completed the Small Business Setup Assistant. Take the rest of the day off”. To trigger display of the completion message, call Assistant.isFinished() .
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.finishedHtml = "Congratulations! You have successfully set up your account.";
...
//Add additional code
```

Assistant.hideAddToShortcutsLink

Property Description	Indicates whether to show or hide the Add to Shortcuts link that appears in the top-right corner of an assistant page. By default, the value is <code>false</code> , which means the Add to Shortcuts link is visible in the UI. If set to <code>true</code> , the Add To Shortcuts link is not visible on an Assistant page.
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.hideAddToShortcutsLink = true;
...
//Add additional code
```

Assistant.hideStepNumber

Property Description	Indicates whether assistant steps are displayed with numbers. By default, the value is <code>false</code> , which means that steps are numbered. If set to <code>true</code> , the assistant does not use step numbers. To change step ordering, set Assistant.isNotOrdered .
-----------------------------	--

Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
assistant.hideStepNumber = true;
...
//Add additional code
```

Assistant.isNotOrdered

Property Description	Indicates whether steps must be completed in a particular sequence. If steps are ordered, users must complete the current step before proceeding to the next step. The default value is <code>false</code> , which means the steps are ordered. Ordered steps appear vertically in the left panel of the assistant. If set to <code>true</code> , steps can be completed in any order. In the UI, unordered steps appear horizontally and below the assistant title
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
```

```

var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
assistant.addStep({
    id : 'idname2',
    label : 'Sample label 2'
});
assistant.isNotOrdered = false;
...
//Add additional code

```

Assistant.title

Property Description	The title for the assistant. The title appears at the top of all assistant pages. This value overrides the title specified in serverWidget.createAssistant(options) .
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var title = assistant.title;
...
//Add additional code

```

serverWidget.AssistantStep

Object Description	Encapsulates a step within a custom NetSuite assistant. Create a step by calling Assistant.addStep(options) . For a complete list of this object's methods and properties, see AssistantStep Object Members .
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .

Module	N/ui/serverWidget Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
  id : 'idname',
  label : 'Sample label'
});
...
//Add additional code
```

AssistantStep.getFieldIds()

Method Description	Gets the IDs for all the fields in a step.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
  id : 'idname',
  label : 'Sample label'
});
assistant.addField({
  id : 'fieldid',
```



```

    type : serverWidget.FieldType.TEXT,
    label : 'Field'
  });
  var fieldIds = assistantStep.getFieldIds();
  ...
  //Add additional code

```

AssistantStep.getSublistFieldIds(options)

Method Description	Gets the IDs for all the sublist fields (line items) in a step.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.group	string	required	The sublist internal ID.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
  id : 'idname',
  label : 'Sample label'
});
var sublist = assistant.addSublist({
  id : 'sublistid',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Editor'
});
var sublistFieldIds = assistantStep.getSublistFieldIds({
  group : 'sublistid'
});
...

```

```
//Add additional code
```

AssistantStep.getLineCount(options)

Method Description	Gets the number of lines on a sublist in a step.  Note: The first line number on a sublist is 0 (not 1).
Returns	The count of line items on a sublist as a number  Note: if the sublist does not exist, -1 is returned.
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.group</code>	string	required	The sublist internal ID.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var sublist = assistant.addSublist({
    id : 'sublistid',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Editor'
});
var numLines = assistantStep.getLineCount({
    group : 'sublistid'
});
...
```

```
//Add additional code
```

AssistantStep.getSubmittedSublistIds()

Method Description	Gets the IDs for all the sublists submitted in a step.
Returns	string[]
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var sublist = assistant.addSublist({
    id : 'sublistid',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Editor'
});
var submittedSublistId = assistantStep.getSubmittedSublistIds();
...
//Add additional code
```

AssistantStep.getSublistValue(options)

Method Description	Gets the current value of a sublist field (line item) in a step.
Returns	The value of a sublist field as a string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.group	string	required	The internal ID of the sublist.	2015.2
options.id	string	required	The internal ID of the sublist field.	2015.2
options.line	number	required	The line number for the sublist field.	2015.2

 **Note:** The first line number on a sublist is 0 (not 1).

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var sublist = assistant.addSublist({
    id : 'sublistid',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Editor'
});
var sublistfield = sublist.addField({
    id : 'fieldid',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var sublistvalue = assistantStep.getSublistValue({
    group: 'sublistid',
    id: 'fieldid',
    line: 0
});
...
//Add additional code
```

AssistantStep.getValue(options)

Method Description	Gets the current value(s) of a field or multi-select field.
Returns	string[]

Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of a field.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var field = assistant.addField({
    id : 'fieldid',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var value = assistantStep.getValue({
    id: 'fieldid'
});
...
//Add additional code
```

AssistantStep.helpText

Property Description	The help text for a step.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code ...
assistantStep.helpText = 'Help Text Goes Here.';
...
//Add additional code
```

AssistantStep.id

Property Description	The internal ID of the step.
Type	string (read-only)
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var id = assistantStep.id;
...
//Add additional code
```

AssistantStep.label

Property Description	The label for the step.
	<div>  Note: To create a label when the step is first added to the assistant, you can use the Assistant.addStep(options) method. </div>
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .

Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var label = assistantStep.label;
...
//Add additional code
```

AssistantStep.stepNumber

Property Description	Indicates where this step appears sequentially in the assistant.
Type	The index of this step as a number. <div> Note: A sequence of assistant steps starts at 1.</div>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
    title : 'Simple Assistant'
});
var assistantStep = assistant.addStep({
    id : 'idname',
    label : 'Sample label'
});
var stepNum = assistantStep.stepNumber;
```

```
...
//Add additional code
```

serverWidget.Button

Object Description	<p>Encapsulates button that appears in a UI object.</p> <p>To add a button, use Form.addButton(options) or Sublist.addButton(options). When adding a button to a record or form, consider using a <code>beforeLoad</code> user event script.</p> <p>Custom buttons only appear during Edit mode. On records, custom buttons appear to the left of the printer icon.</p> <div>  Note: Currently you cannot use SuiteScript to add or remove a custom button to or from the More Actions menu. You can, however, do this using SuiteBuilder point-and-click customization. See the help topic Configuring Buttons and Actions. </div> <p>For a complete list of this object's properties, see Button Object Members.</p>
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var button = form.addButton({
    id : 'buttonid',
    label : 'Test'
});
...

//Add additional code
```

Button.isDisabled

Property Description	<p>Indicates whether a button is grayed-out and disabled.</p> <p>The default value is <code>false</code>.</p> <p>If set to true, the button appears grayed-out in the and cannot be clicked.</p> <div>  Note: This method is not supported for standard NetSuite buttons. This method can be used with custom buttons only. </div>
Type	boolean <code>true</code> <code>false</code>

Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var button = form.addButton({
    id : 'buttonid',
    label : 'Test'
});
button.isDisabled = true;
...
//Add additional code
```

Button.isHidden

Property Description	Indicates whether the button is hidden in the UI. The default value is <code>false</code> , which means the button is visible. If set to <code>true</code> , the button is not visible in the UI. <div> Note: This property is supported on custom buttons and on some standard NetSuite buttons. For a list of supported standard buttons, see the help topic Button IDs. </div>
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
```

```

        title : 'Simple Form'
    });
    var button = form.addButton({
        id : 'buttonid',
        label : 'Test'
    });
    button.isHidden = true;
    ...
    //Add additional code

```

Button.label

Property Description	<p>The label for the button. You can use this property to rename a button based on context, for example to re-label a button for particular users that are viewing a page.</p> <div>  Note: This property is supported on custom buttons and most standard buttons. </div>
Type	string
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var button = form.addButton({
    id : 'buttonid',
    label : 'Test'
});
var label = button.label;
...
//Add additional code

```

serverWidget.Field

Object Description	<p>Encapsulates a body or sublist field. Use fields to record or display information specific to your needs. To add a Field object, use Assistant.addField(options), Form.addField(options), or Sublist.addField(options).</p>
---------------------------	--

	For a complete list of this object's methods and properties, see Field Object Members .
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
...
//Add additional code
```

Field.addSelectOption(options)

Method Description	<p>Adds the select options that appears in the dropdown of a field.</p> <div> Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with <code>Field.addSelectOption(options)</code>. The select values are determined by the source record or list. </div>
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.value	string	required	The internal ID of this select option.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.text	string	required	The label for this select option.	2015.2
options.isSelected	boolean true false	optional	If set to <code>true</code> , this option is selected by default in the UI. The default value for this parameter is <code>false</code> .	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var selectField = form.addField({
    id : 'selectfield',
    type : serverWidget.FieldType.SELECT,
    label : 'Select'
});

selectField.addSelectOption({
    value : '',
    text : ''
});

selectField.addSelectOption({
    value : 'a',
    text : 'Albert'
});
...
//Add additional code
```

Field.getSelectOptions(options)

Method Description	<p>Obtains a list of available options on a select field. The internal ID and label of the options for a select field as name/value pairs is returned. The first 1,000 available options are returned. If you attempt to get select options on a field that is not a select field, or if you reference a field that does not exist on the form, null is returned.</p> <div>  Note: A call to this method may return different results for the same field for different roles. </div>
Returns	Object[]
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Governance	None

Module	N/ui/serverWidget Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.filter	string	optional	A search string to filter the select options that are returned. For example, if there are 50 select options available, and 10 of the options contains 'John', e.g. "John Smith" or "Shauna Johnson", only those 10 options will be returned. Filter values are case insensitive. The filters 'John' and 'john' will return the same select options.	2015.2
options.filteroperator	string	optional	Supported operators are <code>contains</code> <code>is</code> <code>startswith</code> . If not specified, defaults to the <code>contains</code> operator.	2015.2

Syntax

Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var selectField = form.addField({
    id : 'selectfield',
    type : serverWidget.FieldType.SELECT,
    label : 'Select'
});
selectField.addSelectOption({
    value : 'a',
    text : 'Albert'
});
var options = field.getSelectOptions({
    filter : 'a',
    filteroperator: 'startswith'
});
...
//Add additional code
```

Field.setHelpText(options)

Method Description	Sets the help text for the field.
--------------------	-----------------------------------

	When the field label is clicked, a popup displays the help text defined using this method.
Returns	The <code>serverWidget.Field</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.help</code>	string	required	The text in the field help popup.	2015.2
<code>options.showInlineForAssistant</code>	boolean true false	optional	<p>If set to <code>true</code>, the field help will display inline below the field on the assistant, and in a field help popup.</p> <p>The default value is <code>false</code>, which means the field help appears in a popup when the field label is clicked and does not appear inline.</p> <div>  Note: The <code>inline</code> parameter is available only to <code>serverWidget.Field</code> objects that have been added to <code>serverWidget.createAssistant(options)</code> objects. </div>	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var field = form.addField({
  id : 'textfield',
  type : serverWidget.FieldType.TEXT,
  label : 'Text'
});
field.setHelpText({
  help : "Help Text Goes Here."
});
...
```

```
//Add additional code
```

Field.updateBreakType(options)

Method Description	Updates the break type used to add a break in flow layout for the field.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.breakType	serverWidget.FieldBreakType	required	The break type of the field.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.updateBreakType({
    breakType : serverWidget.FieldBreakType.STARTCOL
});
...
//Add additional code
```

Field.updateDisplaySize(options)

Method Description	Updates the width and height of the field.
---------------------------	--

	Only supported on multi-selects, long text, and fields that get rendered as INPUT (type=text) fields. This function is not supported on list/record fields or rich text fields. To set height and width for rich text fields, use Field.richTextWidth and Field.richTextHeight .
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.height	number	required	The new height of the field.	2015.2
options.width	number	required	The new width of the field.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.updateDisplaySize({
    height : 60,
    width : 100
});
...
//Add additional code
```

Field.updateDisplayType(options)

Method Description	Updates the display type for the field.
Returns	serverWidget.Field object
Supported Script Types	Suitelets

	For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.displayType	string	required	The new display type of the field. For more information about possible values, see serverWidget.FieldDisplayType .	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.updateDisplayType({
    displayType : serverWidget.FieldDisplayType.HIDDEN
});
...
//Add additional code
```

Field.updateLayoutType(options)

Method Description	Updates the layout type for the field.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.layoutType	serverWidget.FieldLayoutType	required	The new layout type of the field.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var field = form.addField({
  id : 'textfield',
  type : serverWidget.FieldType.TEXT,
  label : 'Text'
});
field.updateLayoutType({
  layoutType : serverWidget.FieldLayoutType.NORMAL
});
...
//Add additional code
```

Field.alias

Property Description	An alternate name that you can assign to a serverWidget.Field object. By default, the alias is equal to the field's internal ID. This property is only supported on scripted fields created using the N/ui/serverWidget Module .
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```

...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.alias = 'fieldId';
...
//Add additional code

```

Field.defaultValue

Property Description	The default value for this field. If you pass an empty string or any value that is not a number, such as <code>undefined</code> , the field defaults to a blank field in the UI. This property is supported only on scripted fields created using the N/ui/serverWidget Module .
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.defaultValue = 'Insert Text Here.';
...
//Add additional code

```

Field.id

Property Description	The field internal ID.
-----------------------------	------------------------

Type	string (read-only)
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var fieldId = field.id;
...
//Add additional code
```

Field.isMandatory

Property Description	Indicates whether the field is mandatory or optional. If set to <code>true</code> , then the field is defined as mandatory. The default value is <code>false</code> . This property is supported only on scripted fields created using the N/ui/serverWidget Module .
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
```

```

var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.isMandatory = true;
...
//Add additional code

```

Field.label

Property Description	The field label. There is a 40-character limit for custom field labels.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var label = field.label;
...
//Add additional code

```

Field.linkText

Property Description	The text displayed for a link in place of the URL.
Type	string
Supported Script Types	Suitelets

	For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.URL,
    label : 'URL'
});
field.linkText = 'NetSuite';
...
//Add additional code
```

Field.maxLength

Property Description	The maximum length, in characters, of the field (only valid for text, rich text, long text, and textarea fields). This property is supported only on scripted fields created using the N/ui/serverWidget Module .
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
```

```

var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.maxLength = 64;
...
//Add additional code

```

Field.padding

Property Description	The number of empty vertical character spaces above the field. This property is supported only on scripted fields created using the N/ui/serverWidget Module .
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
field.padding = 50;
...
//Add additional code

```

Field.richTextHeight

Property Description	The height of a rich text field, in pixels. The minimum value is 100 pixels and the maximum value is 500 pixels. <div> Note: Rich Text Editing must be enabled.</div>
Type	number

Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.RICHTEXT,
    label : 'Rich Text'
});
field.richTextHeight = 50;
...
//Add additional code
```

Field.richTextWidth

Property Description	The width of a rich text field, in pixels. The minimum value is 250 pixels and the maximum value is 800 pixels. <div> Note: Rich Text Editing must be enabled.</div>
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
```



```

        title : 'Simple Form'
    });
    var field = form.addField({
        id : 'textfield',
        type : serverWidget.FieldType.RICHTEXT,
        label : 'Rich Text'
    });
    field.richTextWidth = 100;
    ...
    //Add additional code

```

Field.type

Property Description	The field type. For example, text, date, currency, select, checkbox etc.
Type	string (read-only)
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var fieldtype = field.type;
...
//Add additional code

```

serverWidget.FieldGroup

Object Description	Encapsulates a field group on serverWidget.createAssistant(options) objects and on serverWidget.Form objects. A field group is a collection of fields that can be displayed in a one or two column format. Assign a field to a field group in order to label, hide or collapse a group of fields. For a complete list of this object's properties, see FieldGroup Object Members .
--------------------	---

Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
...
//Add additional code
```

FieldGroup.isBorderHidden

Property Description	Indicates whether the field group can be collapsed. If set to <code>false</code> , a border around the field group is displayed. The default value is <code>false</code> .
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
fieldgroup.isBorderHidden = true;
...
//Add additional code
```

FieldGroup.isCollapsible

Property Description	Indicates whether the field group can be collapsed. The default value is <code>false</code> , which means the field group displays as a static group that cannot be opened or closed. If set to <code>true</code> , the field group can be collapsed. Only supported for fields on serverWidget.createAssistant(options) objects
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
```

```

        container : 'fieldgroupid'
    });
    fieldgroup.isCollapsible = true;
    ...
    //Add additional code

```

FieldGroup.isCollapsed

Property Description	Indicates whether field group is collapsed or expanded. The default value is <code>false</code> , which means that when the page loads, the field group will not appear collapsed. If set to <code>true</code> , the field group is collapsed. Only supported for fields on serverWidget.createAssistant(options) objects
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
var collapsed = fieldgroup.isCollapsed;
...
//Add additional code

```

FieldGroup.isSingleColumn

Property Description	Indicates whether the field group is aligned. The default value is <code>false</code> .
-----------------------------	--

Type	boolean true false
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
var aligned = fieldgroup.isSingleColumn;
...
//Add additional code
```

FieldGroup.label

Property Description	The label for the field group.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```

...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
var label = fieldgroup.label;
...
//Add additional code

```

serverWidget.Form

Object Description	<p>Encapsulates a NetSuite-looking form After you create a <code>Form</code> object, you can:</p> <ul style="list-style-type: none"> ■ Add a variety of scriptable elements to the form including fields, links, buttons, tabs, and sublists. <p>For a complete list of this object's methods and properties, see Form Object Members.</p>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
...
//Add additional code

```

Form.addButton(options)

Method Description	Adds a button to a form.
---------------------------	--------------------------

Returns	serverWidget.Button object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	optional	The internal ID of the button. If you are adding the button to an existing page, the internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> . For example, if you add a button that appears as Update Order , the button internal ID should be something similar to <code>custpage_updateorder</code> .	2015.2
<code>options.label</code>	string	required	The label for this button.	2015.2
<code>options.functionName</code>	string	optional	The function name to be triggered on a click event.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});

form.addButton({
    id : 'buttonid',
    label : 'Test'
});
...
//Add additional code
```

Form.addCredentialField(options)

Method Description	Adds a text field that lets you store credentials in NetSuite to be used when invoking services provided by third parties. The GUID generated by this field can be reused multiple times until the script executes again. For example, when executing credit card transactions, merchants need to store credentials in NetSuite that are used to communicate with Payment Gateway providers.
---------------------------	---

	<p>The credentials added with this method can be used with the N/ftp Module and the N/https Module.</p> <p>Note the following about this method:</p> <ul style="list-style-type: none"> ■ Credentials associated with this field are stored in encrypted form. ■ No piece of SuiteScript holds a credential in clear text mode. ■ NetSuite reports or forms will never provide to the end user the clear text form of a credential. ■ Any exchange of the clear text version of a credential with a third party must occur over SSL. ■ For no reason will NetSuite ever log the clear text value of a credential (for example, errors, debug message, alerts, system notes, and so on). <div style="border: 1px solid #f0e68c; padding: 5px; margin-top: 10px;">  Important: The default maximum length for a credential field is 32 characters. If needed, use the Field.maxLength property to change this value. </div>
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of the credential field. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the field to an existing page.	2015.2
<code>options.label</code>	string	required	The label for the credential field.	2015.2
<code>options.restrictToDomains</code>	string string[]	required	The domains that the credentials can be sent to, such as <code>'www.mysite.com'</code> . Credentials cannot be sent to a domain that is not specified here. This value can be a domain or a list of domains to which the credentials can be sent.	2015.2
<code>options.restrictToScriptIds</code>	string string[]	required	The IDs of the scripts that are allowed to use this credential field. For example, <code>'customscript_my_script'</code> .	2015.2
<code>options.restrictToCurrentUser</code>	boolean true false	optional	Controls whether use of this credential is restricted to the same user that originally entered the credential. By default, the value is <code>false</code> , which means that multiple users can use the credential. For example, multiple clerks at a store making secure calls to a credit processor using a credential that represents the company they work for. If set to <code>true</code> , the credentials apply to a single user.	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.container</code>	string	optional	The internal ID of the tab or field group to add the credential field to. By default, the field is added to the main section of the form.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete [N/ui/serverWidget Module](#) script example, see [N/ui/serverWidget Module Script Samples](#). For a complete script sample that uses `Form.addCredentialField`, see [N/https Module Script Sample](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var credField = form.addCredentialField({
    id : 'username',
    label : 'Username',
    restrictToDomains : 'www.mysite.com',
    restrictToScriptIds : 'customscript_my_script',
    restrictToCurrentUser : false,
});
credField.maxLength = 64;
...
//Add additional code
```

Form.addField(options)

Method Description	Adds a field to a form.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID of the field. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the field to an existing page. For example, if	2015.2

Parameter	Type	Required / Optional	Description	Since
			you add a field that appears as Purchase Details , the field internal ID should be something similar to <code>custpage_purchasedetails</code> or <code>custpage_purchase_details</code> .	
<code>options.label</code>	string	required	The label for this field.	2015.2
<code>options.type</code>	string	required	<p>The field type for the field. Use the serverWidget.FieldType enum to define the field type.</p> <div>  Important: Long text fields created with SuiteScript have a character limit of 100,000. Long text fields created with Suitebuilder have a character limit of 1,000,000. </div>	2015.2
<code>options.source</code>	string	optional	<p>The internalId or scriptId of the source list for this field if it is a select (List/Record) or multi-select field.</p> <div>  Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with Field.addSelectOption(options). The select values are determined by the source record or list. </div> <div>  Note: For radio fields only, the source parameter must contain the internal ID for the field. For more information about working with radio buttons, see the help topic Working with Radio Buttons. </div>	2015.2
<code>options.container</code>	string	optional	The internal ID of the tab or field group to add the field to. By default, the field is added to the main section of the form.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
...
//Add additional code
```

Form.addFieldGroup(options)

Method Description	Adds a group of fields to a form.
--------------------	-----------------------------------

Returns	serverWidget.FieldGroup object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	An internal ID for the field group.	2015.2
options.label	string	required	The label for this field group.	2015.2
options.tab	string	optional	The internal ID of the tab to add the field group to. By default, the field group is added to the main section of the form.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var fieldgroup = form.addFieldGroup({
    id : 'fieldgroupid',
    label : 'Field Group'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    container : 'fieldgroupid'
});
...
//Add additional code
```

Form.addPageInitMessage(options)

Method Description	Shows a message when users view a record or Suitelet. User event context can be used to control whether the message is shown on records in view, create, or edit mode (not applicable for Suitelets). See context.UserEventType .
---------------------------	---

Returns	Void
Supported Script Types	beforeLoad user event scripts and Suitelets For more information, see beforeLoad(scriptContext) and SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2018.2

Parameters

The options object passed to the `Form.addPageInitMessage(options)` method takes a single property; either a [message.Message](#) object, or the same options object that can be passed to the [message.create\(options\)](#) method. The following tables list the parameters for the previously mentioned object property possibilities.

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.message	message.Message	required	Encapsulates the message to be shown on the form.	2018.2

Parameter	Type	Required / Optional	Description	Since
options.type	message.Type	required	The message type.	2016.1
options.title	string	optional	The message title. This value defaults to an empty string.	2016.1
options.message	string	optional	The content of the message. This value defaults to an empty string.	2016.1
options.duration	int	optional	The amount of time, in milliseconds, to show the message. The default is 0, which shows the message until <code>Message.hide()</code> is called. If you specify a duration for <code>message.create()</code> and <code>message.show()</code> , the value from the <code>message.show()</code> method call takes precedence.	2018.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...

// Options object as parameter
form.addPageInitMessage({type: message.Type.INFORMATION, message: 'Hello world!', duration: 5000});

// Message object as parameter
```

```

var messageObj = message.create({type: message.Type.INFORMATION, message: 'Hello world!', duration: 5000});
form.addPageInitMessage({message: messageObj});

// Show message when the record is in view mode
function beforeLoad(context) {
    if(context.type === 'view')
        context.form.addPageInitMessage(messageOptions);
}
...
//Add additional code

```

Form.addPageLink(options)

Method Description	Adds a link to a form. <div>  Note: You cannot choose where the page link appears. </div>
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	required	The text label for the link.	2015.2
options.type	string	required	The type of page link to add. Use the serverWidget.FormPageLinkType enum to set the value.	2015.2
options.url	string	required	The URL for the link.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addPageLink({
    type : serverWidget.FormPageLinkType.CROSSLINK,

```

```

    title : 'NetSuite',
    url : 'http://www.netsuite.com'
  })
  ...
  //Add additional code

```

Form.addResetButton(options)

Method Description	Adds a reset button to a form. The reset buttons allows a user to clear the entries.
Returns	serverWidget.Button object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.label	string	optional	The label used for this button. If no label is provided, the label defaults to Reset .	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
form.addResetButton({
  label : 'Reset Button'
});
...
//Add additional code

```

Form.addSecretKeyField(options)

Method Description	Adds a secret key field to the form. This key can be used in crypto modules to perform encryption or hashing.
---------------------------	--

	 Important: The default maximum length for a secret key field is 32 characters. If needed, use the Field.maxLength property to change this value.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of the secret key field. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the field to an existing page.	2016.1
options.restrictToScriptIds	string or string[]	required	The script ID of the script that is allowed to use this field.	2016.1
options.restrictToCurrentUser	boolean true false	optional	Controls whether use of this secret key is restricted to the same user that originally entered the key. By default, the value is <code>false</code> , which means that multiple users can use the key. If set to <code>true</code> , the secret key applies to a single user.	2016.1
options.container	string	optional	The internal ID of the tab or field group to add the field to. By default, the field is added to the main section of the form.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete [N/ui/serverWidget Module](#) script example, see [N/ui/serverWidget Module Script Samples](#). For a complete script example that uses `Form.addSecretKeyField(options)`, see [N/crypto Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
skField = form.addSecretKeyField({
  id : 'password',
  restrictToScriptIds : 'customscript_my_script',
  restrictToCurrentUser : false,
});
```

```
skField.maxLength = 64;
...
//Add additional code
```

Form.addSublist(options)

Method Description	Add a sublist to a form. <div>  Note: If the row count exceeds 25, sorting is not supported on static sublists created using this method. </div>
Returns	A <code>serverWidget.Sublist</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.id</code>	string	required	The internal ID name of the sublist. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the sublist to an existing page. For example, if you add a sublist that appears as Purchase Details , the sublist internal ID should be something equivalent to <code>custpage_purchasedetails</code> or <code>custpage_purchase_details</code> .	2015.2
<code>options.label</code>	string	required	The label for this sublist.	2015.2
<code>options.tab</code>	string	optional	The tab under which to display this sublist. If empty, the sublist is added to the main tab.	2015.2
<code>options.type</code>	string	required	The sublist type. Use the serverWidget.SublistType enum to set the value.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
```



```

});
var sublist = form.addSublist({
    id : 'sublistid',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
...
//Add additional code

```

Form.addSubmitButton(options)

Method Description	Adds a submit button to a form. <div>  Note: If the row count exceeds 25, sorting is not supported on static sublists created using this method. </div>
Returns	<code>serverWidget.Button</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description	Since
<code>options.label</code>	string	optional	The label for this button. If no label is provided, the label defaults to "Save".	2016.1

Syntax

 Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see N/ui/serverWidget Module Script Samples .

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addSubmitButton({
    label : 'Submit Button'
});
...

```

```
//Add additional code
```

Form.addSubtab(options)

Method Description	<p>Adds a subtab to a form.</p> <div>  Note: In order for your subtab to appear on your form, there must be at least one object assigned to the subtab. Otherwise, the subtab will not appear. </div> <div>  Note: If you have less than two subtabs on your form, the subtab will not appear. Instead the fields assigned to the tab will appear at the bottom of the form. </div>
Returns	serverWidget.Tab object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the subtab. The internal ID must be in lowercase, contain no spaces. If you are adding the subtab to an existing page, include the prefix <code>custpage</code> . For example, if you add a subtab that appears as Purchase Details , the subtab internal ID should be something similar to <code>custpage_purchasedetails</code> or <code>custpage_purchase_details</code> .	2015.2
options.label	string	required	The label for this subtab.	2015.2
options.tab	string	optional	The tab under which to display this sublist. If empty, the sublist is added to the main tab.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addSubtab({
```

```

    id : 'subtabid',
    label : 'Subtab'
  });
  ...
  //Add additional code

```

Form.addTab(options)

Method Description	Adds a tab to a form. <div>  Note: In order for your tab to appear on your form, there must be at least one object assigned to the tab. Otherwise, the tab will not appear. </div> <div>  Note: If you have less than two tabs on your form, the tab will not appear. Instead the fields assigned to the tab will appear at the bottom of the form. </div>
Returns	serverWidget.Tab object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the tab. The internal ID must be in lowercase and contain no spaces. If you are adding the tab to an existing page, include the prefix <code>custpage</code> . For example, if you add a subtab that appears as Purchase Details , the subtab internal ID should be something similar to <code>custpage_purchasedetails</code> or <code>custpage_purchase_details</code> .	2015.2
options.label	string	required	The label for this tab.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'

```

```
});
var tab = form.addTab({
  id : 'tabid',
  label : 'Tab'
});
...
//Add additional code
```

Form.getButton(options)

Method Description	Returns a Button object by internal ID.
Returns	serverWidget.Button object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the button. Internal IDs must be in lowercase and contain no spaces.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var button = form.addButton({
  id : 'buttonid',
  label : 'Test'
});
var button = form.getButton({
  id : 'buttonid'
});
...
//Add additional code
```

Form.getField(options)

Method Description	Returns a Field object by internal ID.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the field. Internal IDs must be in lowercase and contain no spaces.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var field = form.getField({
    id : 'textfield'
});
...
//Add additional code
```

Form.getSublist(options)

Method Description	Returns a Sublist object by internal ID.
Returns	serverWidget.Sublist object
Supported Script Types	Suitelets

	For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the sublist. Internal IDs must be in lowercase and contain no spaces.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addSublist({
    id : 'sublistid',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
var sublist = form.getSublist({
    id : 'sublistid'
});
...
//Add additional code
```

Form.getSubtab(options)

Method Description	Returns a subtab by internal ID.
Returns	serverWidget.Tab object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the subtab. Internal IDs must be in lowercase and contain no spaces.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
form.addSubtab({
  id : 'subtabid',
  label : 'Subtab'
});
var subtab = form.getSubtab({
  id : 'subtabid'
});
...
//Add additional code
```

Form.getTab(options)

Method Description	Returns a tab object from its internal ID.
Returns	serverWidget.Tab object.
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the tab to retrieve.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addTab({
    id : 'tabid',
    label : 'Tab'
});
var tab = form.getTab({
    id : 'tabid'
});
...
//Add additional code
```

Form.getTabs()

Method Description	Returns an array that contains all the tabs in a form.
Returns	<code>serverWidget.Tab[]</code> objects
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addTab({
    id : 'tabid',
    label : 'Tab'
});
var tabs = form.getTabs();
...
```



```
//Add additional code
```

Form.insertField(options)

Method Description	Inserts a field in front of another field.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.field	serverWidget.Field	required	The Field object to insert.	2016.1
options.nextfield	string	required	The internal ID name of the field you are inserting a field in front of.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field1 = form.addField({
    id : 'textfield1',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
var field2 = form.addField({
    id : 'textfield2',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
form.insertField({
    field : field2,
    nextfield : 'textfield1'
```

```
});
...
//Add additional code
```

Form.insertSublist(options)

Method Description	Inserts a sublist in front of another sublist.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.sublist	serverWidget.Sublist	required	The Sublist object to insert.	2016.1
options.nextsublist	string	required	The internal ID name of the sublist you are inserting a sublist in front of.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var sublist1 = form.addSublist({
  id : 'sublistid1',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Inline Editor Sublist'
});
var sublist2 = form.addSublist({
  id : 'sublistid2',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Inline Editor Sublist'
});
form.insertSublist({
```

```

        sublist : sublist2,
        nextsublist : 'sublistid1'
    });
    ...
    //Add additional code

```

Form.insertSubtab(options)

Method Description	Inserts a subtab in front of another subtab.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.subtab	serverWidget.Tab	required	The Subtab object to insert.	2016.1
options.nextsubtab	string	required	The internal ID name of the subtab you are inserting a subtab in front of.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var subtab1 = form.addSubtab({
    id : 'subtabid1',
    label : 'Subtab'
});
var subtab2 = form.addSubtab({
    id : 'subtabid2',
    label : 'Subtab'
});
form.insertSubtab({

```

```

    subtab : subtab2,
    nextsubtab : 'subtabid1'
  });
  ...
  //Add additional code

```

Form.insertTab(options)

Method Description	Inserts a tab in front of another tab.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.tab	serverWidget.Tab	required	The Tab object to insert.	2016.1
options.nexttab	string	required	The internal ID name of the tab you are inserting a tab in front of.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var tab1 = form.addTab({
  id : 'tabid1',
  label : 'Tab'
});
var tab2 = form.addTab({
  id : 'tabid2',
  label : 'Tab'
});
form.insertTab({
  tab: tab2,

```

```

    nexttab:'tabid1'
  });
  ...
  //Add additional code

```

Form.removeButton(options)

Method Description	Removes a button. This method can be used on custom buttons and certain built-in NetSuite buttons. For more information about built-in NetSuite buttons, see the help topic Button IDs .
Returns	void
Supported Script Types	SuiteScript 2.0 Suitelet Script Type and SuiteScript 2.0 User Event Script Type (beforeLoad(scriptContext) only) For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID name of the button to remove. See the help topic Button IDs . The internal ID must be in lowercase and contain no spaces.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
function beforeLoad(context) {
    var yourForm = context.Form;
    yourForm.removeButton('refund');
}
...
//Add additional code

```

Form.updateDefaultValues(options)

Method Description	Updates the default values of multiple fields on the form.
---------------------------	--

Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
values	object[]	required	An object containing an array of name/value pairs that map field names to field values.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var field = form.addField({
    id : 'textfield',
    type : serverWidget.FieldType.TEXT,
    label : 'Text'
});
form.updateDefaultValues({
    textfield : 'Text Goes Here'
})
...
//Add additional code
```

Form.clientScriptFileId

Property Description	<p>The internal file ID of client script file to be used in this form. Use this property when attaching an on demand client script to a server-side script.</p> <div>  Note: If you deploy a client script to a form using <code>Form.clientScriptFileId</code> or <code>Form.clientScriptModulePath</code>, using the N/log Module adds the logs to the deployment of the parent script. The parent script can be either a <code>beforeLoad</code> user event script or a <code>SuiteScript 2.0 Suitelet Script Type</code>. </div>
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .

Module	N/ui/serverWidget Module
Since	2015.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the Form.clientScriptModulePath property value has already been specified. For more information, see Form.clientScriptModulePath .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.clientScriptFileId = 32;
...
//Add additional code
```

Form.clientScriptModulePath

Property Description	<p>The relative path to the client script file to be used in this form. Use this property when attaching an on demand client script to a server-side script.</p> <div>  Note: If you deploy a client script to a form using Form.clientScriptFileId or Form.clientScriptModulePath, using the N/log Module adds the logs to the deployment of the parent script. The parent script can be either a beforeLoad user event script or a SuiteScript 2.0 Suitelet Script Type. </div>
Type	string
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2016.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the Form.clientScriptFileId property value has already been specified. For more information, see Form.clientScriptFileId .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
objForm.clientScriptModulePath = 'SuiteScripts/formBehavior.js';
...
//Add additional code
```

Form.title

Property Description	The title used for the form.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var title = form.title;
...
//Add additional code
```

serverWidget.List

Object Description	Encapsulates a list. For a complete list of this object's methods and properties, see List Object Members .
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
...
//Add additional code
```

List.addButton(options)

Method Description	Adds a button to a list.
Returns	serverWidget.Button object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of the button. The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the button to an existing page. For example, if you add a button that appears as Update Order , the button internal ID should be something similar to <code>custpage_updateorder</code> .	2015.2
options.label	string	required	The label for this button.	2015.2
options.functionName	string	optional	The function name to call when clicking on this button.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
```

```

var list = serverWidget.createList({
    title : 'Simple List'
});
list.addButton({
    id : 'buttonId',
    label : 'Test'
});
...
//Add additional code

```

List.addColumn(options)

Method Description	Adds a column to a list.
Returns	serverWidget.ListColumn object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID of this column. The internal ID must be in lowercase, contain no spaces.	2015.2
options.label	string	required	The label for this column.	2015.2
options.type	string	required	The field type for this column.  Note: CHECKBOX field types are not supported. For more information about possible values, see serverWidget.FieldType .	2015.2
options.align	string	optional	The default value is <code>left</code> . The layout justification for this column. Possible values include <code>center</code> , <code>right</code> , <code>left</code>	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var list = serverWidget.createList({

```

```

        title : 'Simple List'
    });
    list.addColumn({
        id : 'column1',
        type : serverWidget.FieldType.TEXT,
        label : 'Text',
        align : serverWidget.LayoutJustification.RIGHT
    });
    ...
    //Add additional code

```

List.addEditColumn(options)

Method Description	Adds a column containing Edit or Edit/View links to a Suitelet or Portlet list. These Edit or Edit/View links appear to the left of a previously existing column.
Returns	<code>serverWidget.ListColumn</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.column</code>	object	required	The Edit/View column is added to the left of the column specified here.	2015.2
<code>options.showHrefCol</code>	boolean true false	optional	If set to true, the URL for the link is clickable.	2015.2
<code>options.showView</code>	boolean true false	optional	If <code>true</code> then an Edit/View column will be added. Otherwise only an Edit column will be added.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});

```

```

var columnId = portlet.addColumn({
    id: 'internalId',
    type: 'text',
    label: 'columnLabel'
});

list.addEditColumn({
    column : columnId,
    showView : true
});
...
//Add additional code

```

List.addPageLink(options)

Method Description	Adds a link to a list.
Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	required	The text label for the link.	2015.2
options.type	string	required	The type of page link to add. For more information about possible values, see serverWidget.FormPageLinkType .	2015.2
options.url	string	required	The URL for the link.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
list.addPageLink({
    title : 'NetSuite',

```

```

    type : serverWidget.FormPageLinkType.CROSSLINK,
    url : 'http://www.netsuite.com'
  });
  ...
  //Add additional code

```

List.addRow(options)

Method Description	Adds a single row to a list.
Returns	serverWidget.List
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.row</code>	object	required	A row that consists of either a search.Result, or name/value pairs. Each pair should contain the value for the corresponding Column object in the list.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var list = serverWidget.createList({
  title : 'Simple List'
});
list.addRow({
  row : { columnid1 : 'value1', columnid2 : 'value2' }
});
...
//Add additional code

```

List.addRows(options)

Method Description	Adds multiple rows to a list.
Returns	serverWidget.ListColumn

Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.rows	object[]	required	An array of rows that consist of either a search.Result array, or an array of name/value pairs. Each pair should contain the value for the corresponding Column object in the list.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
list.addRows({
  rows : [{columnid1 : 'value1', columnid2 : 'value2'},
    {columnid1 : 'value2', columnid2 : 'value3'}]
});
...
//Add additional code
```

List.clientScriptFileId

Property Description	The file cabinet ID of client script file to be used in this list.
Type	number
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2016.1

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the List.clientScriptModulePath property value has already been specified. For more information, see List.clientScriptModulePath .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
list.clientScriptFileId = 123;
...
//Add additional code
```

List.clientScriptModulePath

Property Description	The relative path to the client script file to be used in this list.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2016.2

Errors

Error Code	Thrown If
PROPERTY_VALUE_CONFLICT	You attempted to set this value when the List.clientScriptFileId property value has already been specified. For more information, see List.clientScriptFileId .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
objList.clientScriptModulePath = 'SuiteScripts/listBehavior.js';
...
//Add additional code
```

List.style

Property Description	Sets the display style for this list.
----------------------	---------------------------------------

	For more information about possible values, see serverWidget.ListStyle .
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
list.style = serverWidget.ListStyle.REPORT;
...
//Add additional code
```

List.title

Property Description	Sets the list title.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
var title = list.title;
...
//Add additional code
```


serverWidget.ListColumn

Object Description	Encapsulates a list column For a complete list of this object's methods and properties, see ListColumn Object Members .
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
var listcolumn = list.addColumn({
    id : 'column1',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    align : serverWidget.LayoutJustification.RIGHT
});
...
//Add additional code
```

ListColumn.addParamToURL(options)

Method Description	Adds a URL parameter (optionally defined per row) to the list column's URL.
Returns	serverWidget.ListColumn object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.param	string	required	The name for the parameter.

Parameter	Type	Required / Optional	Description
options.value	string	required	The value for the parameter.
options.dynamic	boolean	optional	If true, then the parameter value is actually an alias that is calculated per row.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
var listcolumn = list.addColumn({
    id : 'column1',
    type : serverWidget.FieldType.URL,
    label : 'URL',
});
listcolumn.addParamToURL({
    param : 'index',
    value : '3'
})
...
//Add additional code
```

ListColumn.setURL(options)

Method Description	Sets the base URL for the list column.
Returns	serverWidget.ListColumn
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.1

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.url	string	required	The base URL or a column in the data source that returns the base URL for each row

Parameter	Type	Required / Optional	Description
options.dynamic	boolean	optional	If true, then the URL is actually an alias that is calculated per row.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
var listcolumn = list.addColumn({
    id : 'column1',
    type : serverWidget.FieldType.URL,
    label : 'URL',
});
listcolumn.setURL({
    url : 'http://www.netsuite.com'
})
...
//Add additional code
```

ListColumn.label

Property Description	This list column label.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
var listcolumn = list.addColumn({
```

```

    id : 'column1',
    type : serverWidget.FieldType.URL,
    label : 'URL',
  });
  var label = listcolumn.label;
  ...
  //Add additional code

```

serverWidget.Sublist

Object Description	<p>Encapsulates a sublist on a serverWidget.Form or an serverWidget.createAssistant(options) object.</p> <p>To add a sublist, use Assistant.addSublist(options) or Form.addSublist(options).</p> <div>  Note: This object is read-only except for instances created via the serverWidget module using Suitelets or beforeLoad user event scripts. </div> <p>For a complete list of this object's methods and properties, see Sublist Object Members.</p>
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var sublist = form.addSublist({
  id : 'sublist',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Inline Editor Sublist'
});
...
//Add additional code

```

Sublist.addButton(options)

Method Description	Adds a button to a sublist.
Returns	serverWidget.Button
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>

Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The internal ID name of the button. The internal ID must be in lowercase and without spaces.
options.label	string	required	The label for the button.
options.functionName	string	optional	The function name to be triggered on a button click.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.addButton({
    id : 'buttonId',
    label : 'Test'
});
...
//Add additional code
```

Sublist.addField(options)

Method Description	Adds a field to a sublist.
Returns	serverWidget.Field object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None

Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	The internal ID for this field. The internal ID must be in lowercase and without spaces.	2015.2
options.label	string	required	The label for this field.	2015.2
options.type	string	required	The field type. Use the serverWidget.FieldType enum to set this value. The <code>INLINEHTML</code> and <code>value</code> is not supported with this method. The <code>MULTISELECT</code> value is not supported for SuiteScript 2.0 Suitelets.  Note: If you have set the <code>type</code> parameter to <code>SELECT</code> , and you want to add custom options to the select field, you must set <code>source</code> to <code>NULL</code> .	2015.2
options.source	string	optional	The internalId or scriptId of the source list for this field. Use this parameter if you are adding a select (List/Record) type of field.  Note: If you want to add custom options on a select field, you must set the source parameter to <code>NULL</code> .  Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with Field.addSelectOption(options) . The select values are determined by the source record or list.	2015.2

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.addField({
    id : 'fieldId',
```

```

    type : serverWidget.FieldType.DATE,
    label : 'Date'
  });
  ...
  //Add additional code

```

Sublist.addMarkAllButtons()

Method Description	Adds a Mark All and an Unmark All button to a <code>LIST</code> type of sublist.
Returns	A <code>serverWidget.Button[]</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var sublist = form.addSublist({
  id : 'sublist',
  type : serverWidget.SublistType.LIST,
  label : 'List Sublist'
});
sublist.addMarkAllButtons();
...
//Add additional code

```

Sublist.addRefreshButton()

Method Description	Adds a Refresh button to a <code>LIST</code> type of sublist.
Returns	<code>serverWidget.Button</code> object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.LIST,
    label : 'List Sublist'
});
sublist.addRefreshButton();
...
//Add additional code
```

Sublist.getField(options)

Method Description	Returns a Field object on a sublist.
Returns	serverWidget.Field
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2016.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.id</code>	string	required	The field internal ID (for example, use item as the ID for the Item field). For more information about supported sublists, internal IDs, and field IDs, see the SuiteScript Records Browser .

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```



```

...
var itemField = form.getSublist({id: 'item'}).getField({id: 'item'});
...
//Add additional code

```

Sublist.getSublistValue(options)

Method Description	Gets a field value on a sublist.
Returns	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The internal ID of a field.
options.line	number	required	The line number for this field.

 **Note:** The first line number on a sublist is 0 (not 1).

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var sublistvalue = sublist.getSublistValue({
  id : 'quantity',
  line: 1
})
...
//Add additional code

```

Sublist.setSublistValue(options)

Method Description	Sets the value of a sublist field.
---------------------------	------------------------------------

Returns	void
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The internal ID name of the line item field being set.
options.line	number	required	The line number for this field. <div> Note: The first line number on a sublist is 0 (not 1).</div>
options.value	string	required	The value for the field being set.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});

sublist.addField({
    id : 'sublist',
    type: ui.FieldType.TEXT,
    label: 'Text Field'
});

sublist.setSublistValue({
    id : 'sublist',
    line : 2,
    value : "Text"
});
...
```

```
//Add additional code
```

Sublist.updateTotallingFieldId(options)

Method Description	Updates the ID of a field designated as a totalling column, which is used to calculate and display a running total for the sublist.
Returns	serverWidget.Sublist object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.id</code>	string	required	The internal ID name of the field to use as a total field.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
sublist.updateTotallingFieldId({
    id : 'fieldId'
})
...
//Add additional code
```

Sublist.updateUniqueFieldId(options)

Method Description	Updates a field ID that is to have unique values across the rows in the sublist.
	 Note: This method is available on inlineeditor and editor sublists only.
Returns	serverWidget.Sublist object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None

Module	N/ui/serverWidget Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.id	string	required	The internal ID name of the field to use as a unique field.

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.addField({
    id : 'fieldId',
    type : serverWidget.FieldType.DATE,
    label : 'Date'
});
sublist.updateUniqueFieldId({
    id : 'fieldId'
})
...
//Add additional code
```

Sublist.displayType

Property Description	The display style for a sublist. Use the serverWidget.SublistDisplayType enum to set this value.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.displayType = serverWidget.SublistDisplayType.HIDDEN;
...
//Add additional code
```

Sublist.helpText

Property Description	The inline help text for a sublist.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.helpText = "Help Text Goes Here.";
...
//Add additional code
```

Sublist.label

Property Description	The label for this sublist.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var sublist = form.addSublist({
  id : 'sublist',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Inline Editor Sublist'
});
var label = sublist.label;
...
//Add additional code
```

Sublist.lineCount

Property Description	The number of line items on a sublist. <div> Note: The first line number on a sublist is 0 (not 1).</div>
Type	number (read-only)
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```

...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
var numLines = sublist.lineCount;
...
//Add additional code

```

serverWidget.Tab

Object Description	<p>Encapsulates a tab or subtab on a serverWidget.Form object. You can add a new tab or subtab to a form using one of the following methods:</p> <ul style="list-style-type: none"> ■ Form.addSubtab(options) ■ Form.addTab(options) ■ Form.insertSubtab(options) ■ Form.insertTab(options) <p>The internal ID must be in lowercase, contain no spaces, and include the prefix <code>custpage</code> if you are adding the field to an existing page.</p> <div>  Note: In order for your tab to appear on your form, there must be at least one object assigned to the tab. Otherwise, the tab will not appear. </div> <div>  Note: If you have less than two tabs on your form, the tab will not appear. Instead the fields assigned to the tab will appear at the bottom of the form. </div> <p>For a complete list of this object's properties, see Tab Object Members.</p>
Supported Script Types	<p>Suitelets</p> <p>For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var tab = form.addTab({

```

```

        id : 'tabid1',
        label : 'Tab 1'
    });

    var tab = form.addTab({
        id : 'tabid2',
        label : 'Tab 2'
    });

    form.addField({
        id : 'tabid1',
        type: ui.FieldType.TEXT,
        label: 'Tab 1 Field'
    });

    form.addField({
        id : 'tabid2',
        type: ui.FieldType.TEXT,
        label: 'Tab 2 Field'
    });...
//Add additional code

```

Tab.helpText

Property Description	The inline help text for a tab or subtab.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var tab = form.addTab({
    id : 'tabid',
    label : 'Tab'
});
tab.helpText = 'Help Text Goes Here';
...
//Add additional code

```


Tab.label

Property Description	The label for a tab or subtab.
Type	string
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var tab = form.addTab({
  id : 'tabid',
  label : 'Tab'
});
var label = tab.label;
...
//Add additional code
```

serverWidget.createAssistant(options)

Method Description	Creates an assistant object.
Returns	serverWidget.Assistant object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.title	string	required	The title of the assistant. This title appears at the top of all assistant pages.	2015.2

Parameter	Type	Required / Optional	Description	Since
<code>options.hideNavBar</code>	boolean <code>true</code> <code>false</code>	optional	Indicates whether to hide the navigation bar menu. By default, set to <code>false</code> . The header appears in the top-right corner on the assistant. If set to <code>true</code> , the header on the assistant is hidden from view.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
...
//Add additional code
```

serverWidget.createForm(options)

Method Description	Creates a form object.
Returns	serverWidget.Form object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.title</code>	string	required	The title of the form.	2015.2
<code>options.hideNavBar</code>	boolean <code>true</code> <code>false</code>	optional	Indicates whether to hide the navigation bar menu. By default, set to <code>false</code> . The header appears in the top-right corner on the form. If set to <code>true</code> , the header on the assistant is hidden from view.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code ...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
...
//Add additional code
```

serverWidget.createList(options)

Method Description	Instantiates a standalone list.
Returns	serverWidget.List object
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Governance	None
Module	N/ui/serverWidget Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
<code>options.title</code>	string	required	The title of the list.	2016.1
<code>options.hideNavBar</code>	boolean true false	optional	Indicates whether to hide the navigation bar menu. By default, set to <code>false</code> . The header appears in the top-right corner on the form. If set to <code>true</code> , the header on the assistant is hidden from view.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
```

```
title : 'Simple List'
});
...
//Add additional code
```

serverWidget.AssistantSubmitAction

Enum Description	<p>Holds the string values for submit actions performed by the user. This enum is used to set the value of the Assistant.getLastAction().</p> <p>After a <code>finish</code> action is submitted, by default, the text “Congratulations! You have completed the <assistant title>” appears on the finish page.</p> <p>In a non-sequential process (steps are unordered), <code>jump</code> is used to move to the user’s last action.</p> <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</div>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Values

- BACK
- CANCEL
- FINISH
- JUMP
- NEXT

Syntax

The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var assistant = serverWidget.createAssistant({
  title : 'Simple Assistant'
});
...
if (assistant.getLastAction() == serverWidget.AssistantSubmitAction.CANCEL) {
  ...
}
...
//Add additional code
```

serverWidget.FieldBreakType

Enum Description	<p>Enumeration that holds the string values for supported field break types. This enum is used to set the value of the <code>breakType</code> parameter when <code>Field.updateBreakType(options)</code> is called.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

NONE	This is the default value for field break type.
STARTCOL	This value moves the field into a new column. Additionally, it disables automatic field balancing if set on any field.
STARTROW	<p>This value places a field located outside of a field group on a new row. This value only works on fields with a Field Layout Type set to OUTSIDE, OUTSIDEABOVE or OUTSIDEBELOW.</p> <p>For more information, see serverWidget.FieldLayoutType and Field.updateLayoutType(options).</p>

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var field = form.addField({
  id : 'textfield',
  type : serverWidget.FieldType.TEXT,
  label : 'Text'
});

field.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.OUTSIDE
});

field.updateBreakType({
  breakType : serverWidget.FieldBreakType.STARTROW
});
...
//Add additional code
```

serverWidget.FieldDisplayType

Enum Description	<p>Enumeration that holds the string values for supported field display types. This enum is used to set the value of the <code>displayType</code> parameter when <code>Field.updateDisplayType(options)</code> is called.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

Value	Description of Field Type
DISABLED	Prevents a user from changing the field
ENTRY	The sublist field appears as a data entry input field (for a select field without a checkbox)
HIDDEN	The field on the form is hidden.
INLINE	The field appears as inline text
NORMAL	The field appears as a normal input field (for non-sublist fields)
READONLY	The field is disabled but it is still selectable and scrollable (for textarea fields)

Syntax



Important: The following code snippets show the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var field = form.addField({
  id : 'textfield',
  type : serverWidget.FieldType.TEXT,
  label : 'Text'
});
field.updateDisplayType({
  displayType: serverWidget.FieldDisplayType.HIDDEN
});
...
//Add additional code
```

serverWidget.FieldLayoutType

Enum Description	<p>Enumeration that holds the string values for the supported types of field layouts. This enum is used to set the value of the <code>layoutType</code> parameter when <code>Field.updateLayoutType(options)</code> is called.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

STARTROW	This value makes the field appear first in a horizontally aligned field group in the normal field layout.
MIDROW	This value makes the field appear in the middle of a horizontally aligned field group in the normal field layout.
ENDROW	This value makes the field appear last in a horizontally aligned field group in the normal field layout.
OUTSIDE	This value makes the field appear outside (above or below based on form default) the normal field layout area.
OUTSIDEBELOW	This value makes the field appear below the normal field layout area. Using this allows you to position a field below a field group.
OUTSIDEABOVE	This value makes the field appear above the normal field layout area. Using this allows you to position a field above a field group.
NORMAL	This value makes the fields appear in its default position.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title: 'Simple Form'
});
var field = form.addField({
  id: 'textfield',
  type: serverWidget.FieldType.TEXT,
  label: 'Text'
});
field.updateLayoutType({
```

```
layoutType: serverWidget.FieldLayoutType.OUTSIDE BELOW
});
...
//Add additional code
```

serverWidget.FieldType

Enum Description	<p>Enumeration that holds the values for supported field types. This enum is used to set the value of the <code>type</code> parameter when <code>Form.addField(options)</code> is called.</p> <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.</div> <div> Important: Long text fields created with SuiteScript have a character limit of 100,000. Long text fields created with Suitebuilder have a character limit of 1,000,000.</div>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Values

<ul style="list-style-type: none">CHECKBOXCURRENCYDATEDATETIMEDATETIMETZEMAILFILEFLOATHELPINLINEHTMLINTEGERIMAGELABEL	<ul style="list-style-type: none">LONGTEXTMULTISELECTPASSWORDPERCENTPHONESELECTRADIORICHTEXTTEXTTEXTAREATIMEOFDAYURL
---	---

Consider the following as you work with these field types:

- The FILE field type is available only for Suitelets and will appear on the main tab of the Suitelet page. FILE fields cannot be added to tabs, subtabs, sublists, or field groups and are not allowed on existing pages.
- The INLINEHTML field type is not supported with `Sublist.addField(options)`.
- The IMAGE field type is available only for fields that appear on list/staticlist sublists. You cannot specify an IMAGE field on a form.

- The MULTISELECT field type is not supported by SuiteScript 2.0 Suitelets.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
  title : 'Simple Form'
});
var field = form.addField({
  id : 'textfield',
  type : serverWidget.FieldType.TEXT,
  label : 'Text'
});
...
//Add additional code
```

serverWidget.FormPageLinkType

Enum Description	<p>Enumeration that holds the string values for supported page link types on a form. This enum is used to set the value of the <code>type</code> parameter when <code>Form.addPageLink(options)</code> is called.</p> <div> Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

Value	Notes
BREADCRUMB	Link appears on the top-left corner after the system bread crumbs
CROSSLINK	Link appears on the top-right corner

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
```

```

...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
form.addPageLink({
    type : serverWidget.FormPageLinkType.CROSSLINK,
    title : 'NetSuite',
    url : 'http://www.netsuite.com'
})
...
//Add additional code

```

serverWidget.LayoutJustification

Enum Description	<p>Enumeration that holds the string values for supported justification layouts. This enum is used to set the value of the <code>align</code> parameter when <code>List.addColumn(options)</code> is called.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

- CENTER
- LEFT
- RIGHT

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```

//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
list.addColumn({
    id : 'column1',
    type : serverWidget.FieldType.TEXT,
    label : 'Text',
    align : serverWidget.LayoutJustification.RIGHT

```

```
});
...
//Add additional code
```

serverWidget.ListStyle

Enum Description	<p>Enumeration that holds the string values for supported list styles. This enum is used to set the value of the List.style property.</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	<p>Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type.</p>
Module	N/ui/serverWidget Module
Since	2015.2

Values

- GRID
- REPORT
- PLAIN
- NORMAL

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var list = serverWidget.createList({
    title : 'Simple List'
});
list.style = serverWidget.ListStyle.REPORT;
...
//Add additional code
```

serverWidget.SublistDisplayType

Enum Description	<p>Enumeration that holds the string values for supported sublist display types. This enum is used to set the value of the Sublist.displayType property.</p>
-------------------------	--

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module
Since	2015.2

Values

 HIDDEN
 NORMAL

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
sublist.displayType = serverWidget.SublistDisplayType.HIDDEN;
...
//Add additional code
```

serverWidget.SublistType

Enum Description	<p>Enumeration that holds the string values for valid sublist types. This enum is used to define the <code>type</code> parameter when <code>Form.addSublist(options)</code> is called</p> <div>  Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value. </div>
Supported Script Types	Suitelets For more information, see SuiteScript 2.0 Suitelet Script Type .
Module	N/ui/serverWidget Module

Since	2015.2
-------	--------

Values

Value	Description
INLINEEDITOR EDITOR	<p>These types of sublists are both fully editable. The only difference between these types is their appearance in the UI:</p> <ul style="list-style-type: none"> With an inline editor sublist, a new line is displayed at the bottom of the list after existing lines. To add a line, a user working in the UI clicks inside the new line and adds a value to each column as appropriate. Examples of this style include the Item sublist on the sales order record and the Expense sublist on the expense report record. With an editor sublist, a user in the UI adds a new line by working with fields that are displayed above the existing sublist lines. This style is not common on standard NetSuite record types.
LIST	<p>This type of sublist has a fixed number of lines. You can update an existing line, but you cannot add lines to it.</p> <div>  Note: To make a field within a LIST type sublist editable, use <code>Field.updateDisplayType(options)</code> and the enum <code>serverWidget.FieldDisplayType</code> to update the field display type to ENTRY. </div>
STATICLIST	This type of sublist is read-only. It cannot be edited in the UI, and it is not available for scripting.

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/ui/serverWidget Module Script Samples](#).

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form with Inline Editor type Sublist'
});
var sublist = form.addSublist({
    id : 'sublist',
    type : serverWidget.SublistType.INLINEEDITOR,
    label : 'Inline Editor Sublist'
});
...
//Add additional code
```

The following code snippet shows how to make a field within a LIST type sublist editable by updating the fieldDisplayType to ENTRY.

```
//Add additional code
...
var form = serverWidget.createForm({
    title : 'Simple Form with List type Sublist'
});
var sublist = form.addSublist({
```

```

    id : 'sublist',
    type : serverWidget.SublistType.LIST,
    label : 'List Type Sublist'
  });
  var internalId = sublist.addField({
    id : 'id',
    label : 'Internal ID',
    type : serverWidget.FieldType.TEXT
  });
  internalId.updateDisplayType({displayType: serverWidget.FieldDisplayType.ENTRY});
  ...
  //Add additional code

```

N/url Module

Use the url module to determine URL navigation paths within NetSuite and format URL strings.

- [N/url Module Members](#)
- [N/url Module Script Samples](#)

N/url Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	url.format(options)	string	Server-side scripts	Converts (serializes) URL query parameters into a string.
	url.resolveDomain(options)	string	Client and server-side scripts	Returns a domain name for a NetSuite account.
	url.resolveRecord(options)	string	Server-side scripts	Returns an internal URL string to a NetSuite record.
	url.resolveScript(options)	string	Server-side scripts	Returns an external or internal URL string to a script.
	url.resolveTaskLink(options)	string	Server-side scripts	Returns an internal URL for a tasklink.
Enum	url.HostType	enum	Server-side scripts	An enum used to populate the <code>hostType</code> parameter of the url.resolveDomain(options) method.

N/url Module Script Samples

The following script samples show how to use the url module.

These samples use the `require` function, so that you can copy each script into the debugger and test it, after making any necessary edits. Remember that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).



Important: Some values in these samples are placeholders. Before using these samples, replace these values with valid ones from your NetSuite account. If you run a script with an invalid value, the system may throw an error.

The following example retrieves the relative URL of a record. With the internal ID value used in this example, the returned output would be `/app/accounting/transactions/salesord.nl?id=6&e=T&compid=`, followed by the NetSuite account ID.

```
/**
 * @NApiVersion 2.x
 */
require(['N/url'],
  function(url) {
    var output = url.resolveRecord({
      recordType: 'salesorder',
      recordId: 6,
      isEditMode: true
    });
  });
```

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

```
/**
 * @NApiVersion 2.x
 */
// This example shows how to get the absolute url of a record.
// Company context is required to run this client's script.

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 output = scheme + host + relativePath;
    }
    resolveRecordUrl();
  }
);
```

The following example shows how to get the domain for calling a RESTlet.

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

```

        var sCompId = 'MSTRWLF';
        var output = url.resolveDomain({
            hostType: url.HostType.RESTLET,
            accountId: sCompId
        });
    }
    resolveDomainUrl();
}
);

```

The following example creates a URL and then does a secure HTTPS POST request to that URL with an empty body. The server's response is logged.

```

/**
 * @NApiVersion 2.x
 */
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());
    }
});

```

url.format(options)

Method Description	Creates a serialized representation of an object containing query parameters. Use the returned value to build a URL query string.
Returns	URL as a string
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/url Module
Since	2015.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.domain	string	required	The domain name.	2015.1

Parameter	Type	Required / Optional	Description	Since
options.params	Object	required	Additional URL parameters as name/value pairs.	2015.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module Script Samples](#).

For a script that uses the following code snippet, the returned output is `http://fruitland.com?fruit=grape&seedless=true&variety=Concord+Giant&PLU=4272`, expressed as a string.

```
//Add additional code
...
var output = url.format({
  domain: 'http://fruitland.com',
  params: {
    fruit: 'grape',
    seedless: true,
    variety: 'Concord Giant',
    PLU: 4272
  }
});
...
//Add additional code
```

url.resolveDomain(options)

Method Description	Returns a domain name for a NetSuite account.
Returns	string
Supported Script Types	Client and server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/url Module
Since	2017.1

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.hostType	string	required	The type of domain name you want to retrieve. Set this value using the url.HostType enum.	2017.1
options.accountId	string	optional	The NetSuite account ID for which you want to retrieve data. If no account is specified, the system returns data on the account that is running the script.	2017.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module Script Samples](#).

```
//Add additional code
...
var output = url.resolveDomain({
    hostType: url.HostType.APPLICATION,
    accountId: '012345'
});
...
//Add additional code
```

url.resolveRecord(options)

Method Description	Returns the URL string to a NetSuite record.
Returns	URL to a NetSuite record as a string
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/url Module
Since	2015.1

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.recordType	string	required	The type of record. For example, 'transaction'.	2015.1
options.recordId	string	required	The internal ID of the target record instance.	2015.1
options.isEditMode	boolean true false	required	If set to <code>true</code> , returns a URL for the record in Edit mode. If set to <code>false</code> , returns a URL for the record in View mode The default value is <code>View</code> .	2015.1
options.params	Object	optional	Object used to add parameters for a custom URL. For example, a query to a database or to a search engine	2015.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module Script Samples](#).

```
//Add additional code
```

```

...
var output = url.resolveRecord({
  recordType: 'salesorder',
  recordId: 6,
  isEditMode: true
});
...
//Add additional code

```

url.resolveScript(options)

Method Description	Returns an external or internal URL string to a script.
Returns	The URL as a string
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/url Module
Since	2015.1

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.scriptId	string	required	The internal ID of the script. The ID must identify a RESTlet or a Suitelet.	2015.1
options.deploymentId	string	required	The internal ID of the deployment script	2015.1
options.returnExternalUrl	boolean true false	required	Indicates whether to return the External URL. By default, the internal URL is returned.	2015.1
options.params	Object	optional	The object containing name/value pairs to describe the query.	2015.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module Script Samples](#).

```

//Add additional code
...
var output = url.resolveScript({
  scriptId: 'custom_script',
  deploymentId: 'custom_script_deployment',
  returnExternalUrl: true
});

```

```
...
//Add additional code
```

url.resolveTaskLink(options)

Method Description	Returns the internal URL to a NetSuite tasklink.
Returns	The URL as a string
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/url Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.id	string	required	Internal ID for the tasklink.  Note: Each page in NetSuite has a unique Tasklink Id associated with it for a specific record type. You can determine the Tasklink for a page within NetSuite by viewing the HTML page source. Search for a string similar to the following, where LIST_SCRIPT refers to the TASKLINK: onclick="nlPopupHelp('LIST_SCRIPT','help').	2015.1
options.params	Map	optional	The Map object containing name/value pairs to describe the query.	2015.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module Script Samples](#).

```
//Add additional code
...
u = url.resolveTaskLink('SRCH_JOB', p);
...
//Add additional code
```

url.HostType

Enum Description	Enumeration whose string values each describe a category of domain name. This enum is used to set the value of the hostType parameter of the url.resolveDomain(options) method.
-------------------------	---

	 Note: JavaScript does not include an enumeration type. The SuiteScript 2.0 documentation utilizes the term enumeration (or enum) to describe the following: a plain JavaScript object with a flat, map-like structure. Within this object, each key points to a read-only string value.
Type	enum
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types .
Module	N/url Module
Since	2017.1

Values

Value	Description	Sample Result
APPLICATION	The domain for UI access, for users with a non-Customer Center role.	system.na2.netsuite.com
CUSTOMER_CENTER	The domain for UI access, for users with a Customer Center role.	system.na2.netsuite.com
FORM	The domain for forms hosted online, usually in Suitelets.	forms.na2.netsuite.com
RESTLET	The domain for calling a RESTlet from an external source.	rest.na2.netsuite.com
SUITETALK	The domain for SuiteTalk (web services) requests.	webservices.na2.netsuite.com

 **Warning:** The results returned, as shown in the sample results column, may change without notice. Because these values can change, your scripts must dynamically discover domain names. For more details, see the help topic [Understanding Multiple Data Centers](#).

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/url Module](#).

```
//Add additional code
...
var output = url.resolveDomain({
  hostType: url.HostType.APPLICATION,
  accountId: '012345'
});
...
//Add additional code
```

N/util Module

This module exposes the util Object and its members, made up primarily of methods that verify type on objects and primitives in a SuiteScript 2.0 script.

Each type verification method (for example, `util.isArray(obj)`) returns a boolean value, based on evaluation of the `obj` parameter.

If you need to identify a type specific to SuiteScript 2.0, use the `toString()` global method.

Note: The `util` Object can be accessed globally or by loading this module. Load the `N/util` module when you want to manually access the `util` module members, such as for testing purposes. For more information about global objects, see [SuiteScript 2.0 Global Objects](#).

- [N/util Module Members](#)
- [N/util Module Script Sample](#)

N/util Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>util.isArray(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript and false otherwise.
	<code>util.isBoolean(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a Boolean and false otherwise.
	<code>util.isDate(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript Date object and false otherwise.
	<code>util.each(iterable, callback)</code>	Object or Array	Client and server-side scripts	Iterates over each member in an Object or Array.
	<code>util.extend(receiver, contributor)</code>	Object	Client and server-side scripts	Copies the properties in a source object to a destination object and returns the destination object.
	<code>util.isFunction(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript Function object and false otherwise.
	<code>util.isNumber(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript Number object or a value that evaluates to a Number object, and false otherwise.
	<code>util.isObject(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a strictly a JavaScript Object, and false otherwise.
	<code>util.isRegExp(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript RegExp object or a value that evaluates to a RegExp object, and false otherwise.
	<code>util.isString(obj)</code>	boolean true false	Client and server-side scripts	Returns true if the <code>obj</code> parameter is a JavaScript String object or a value that evaluates to a String object, and false otherwise.
	<code>util.nanoTime()</code>	number	Server-side scripts	Returns the amount of time elapsed from an arbitrary fixed point, in nanoseconds.
	<code>util.each(iterable, callback)</code>	Object or Array	Client and server-side scripts	Iterates over each member in an Object or Array.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>util.extend(receiver, contributor)</code>	Object	Client and server-side scripts	Copies the properties in a source object to a destination object.

N/util Module Script Sample

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

```
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');
    });
    // log.debug(rec) //Shows the JSON representation of the current values in a record object
    var id = rec.save();
});
```



Warning: This script sample includes hard-coded values for the purpose of illustration. To run this sample in the SuiteScript debugger, you must replace these hard-coded values with values from records in your account. For information about debugging, see the help topic [Using the SuiteScript Debugger](#).

util.isArray(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript Array object and false otherwise.
Returns	boolean true false

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Global object	util Object
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax

 **Important:** The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
var records = ["Sales Order", "Invoice", "Item Fulfillment"];
util.isArray(records); // returns true

var record = "Sales Order";
util.isArray(record); // returns false
...
//Add additional code
```

util.isBoolean(obj)

Method Description	Returns true if the <code>obj</code> parameter is a boolean and false otherwise.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
var flag = true;
util.isBoolean(flag); // returns true
util.Boolean(true);   // returns true

util.Boolean(1);      // returns false
...
//Add additional code
```

util.isDate(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript Date object and false otherwise.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
var todaysDate = new Date();
util.isDate(todaysDate); // returns true
util.isDate(new Date()); // returns true

var today = "September 28, 2015";
util.isDate(today);      // returns false
...
//Add additional code
```

```
//Add additional code
```

util.isFunction(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript Function object and false otherwise.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
<code>obj</code>	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
function test() {};
var test2 = function() {};

util.isFunction(test); // returns true
util.isFunction(test2); // returns true
...
//Add additional code
```

util.isNumber(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript Number object or primitive, and false otherwise.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
util.isNumber(112);           // returns true
util.isNumber("112");        // returns false
util.isNumber(NaN);          // returns true

var testNum = 112;
util.isNumber(testNum.valueOf()); // returns true
...
//Add additional code
```

util.isObject(obj)

Method Description	Returns true if the <code>obj</code> parameter is a plain JavaScript object(<code>new Object()</code> or <code>{}</code> for example), and false otherwise. Use this method, for example, to verify that a variable is a JavaScript object and not a JavaScript Function.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
```

```

...
util.isObject({});           // returns true
util.isObject(function() {}); // returns false
...
//Add additional code

```

util.isRegExp(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript RegExp object, and false otherwise.
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```

//Add additional code
...
util.isRegExp(/this is a regexp/);           // returns true
util.isRegExp(new RegExp('this is another regexp')); // returns true
...
//Add additional code

```

util.isString(obj)

Method Description	Returns true if the <code>obj</code> parameter is a JavaScript String object or primitive, and false otherwise
Returns	boolean true false
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module

Since	2016.1
-------	--------

Parameters

Parameter	Type	Required / Optional	Description	Since
obj	Object Primitive	Required	Object for which you want to verify the type.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
...
util.isString('');           // returns true
util.isString('a string');   // returns true

var myString = new String('another string');
util.isString(myString);     // returns true

util.isString(null);         // returns false
...
//Add additional code
```

util.nanoTime()

Method Description	Returns the current time (epoch) in nanoseconds. You can use this method to measure elapsed time between two events.
Returns	string
Supported Script Types	Server-side scripts For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. It demonstrates how to calculate the number of nanoseconds between two calls to `util.nanoTime()`. For a full script sample, see [N/util Module Script Sample](#).

```
//Add additional code
var startTime = util.nanoTime();
...
var elapsedTime = util.nanoTime() - startTime;
...
//Add additional code
```

util.each(iterable, callback)

Method Description	Iterates over each member in an Object or Array. This method calls the <code>callback</code> function on each member of the <code>iterable</code> .
Returns	The original collection as an Object Array
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module
Since	2016.1

Parameters

Parameter	Type	Required / Optional	Description	Since
iterable	Object Array	Required	The data collection to iterate on	2016.1
callback	Function	Required	Takes the custom logic that you want to execute on each member of your collection of data.	2016.1

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

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

util.extend(receiver, contributor)

Method Description	Method used to copy the properties in a source object to a destination object. Returns the destination object. You can use this method to merge two objects.
Returns	The Object receiving the properties copied from the contributor
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/util Module

Since	2016.1
-------	--------

Syntax



Important: The following code snippets shows the syntax for this member. It is not a functional example. For a full script sample, see [N/util Module Script Sample](#).

This snippet shows combining two objects without the same keys:

```
//Add additional code
...
var colors = {};
var firstSet = {'color1':'red',
               'color2':'yellow',
               'color3':'blue'};
var secondSet = {'color4':'green',
                'color5':'orange',
                'color6':'violet'};

// Extends colors object with the information in firstSet
// Colors will get {'color1':'red','color2':'yellow','color3':'blue'}
util.extend(colors, firstSet);

// Extends colors object with the information in secondSet
// Colors will get
{'color1':'red','color2':'yellow','color3':'blue','color4':'green','color5':'orange','color6':'violet'}
util.extend(colors, secondSet);
});
...
//Add additional code
```

The following snippet shows overriding two objects with a few similar keys:

```
//Add additional code
...
var colors = {};
var firstSet = {'color1':'red',
               'color2':'yellow',
               'color3':'blue'};
};
var secondSet = {'color2':'green',
                'color3':'orange',
                'color4':'violet'};

// Extends colors object with the information in firstSet
// Colors will get {'color1':'red','color2':'yellow','color3':'blue'}
util.extend(colors, firstSet);

// Extends colors object with the information in secondSet and overrides the value if there are similar keys
// Colors will get {'color1':'red','color2':'green','color3':'orange','color4':'violet'}
util.extend(colors, secondSet);
```

```

var x = 0;
});
...
//Add additional code

```

N/workflow Module

This module loads the workflow module to initiate new workflow instances or trigger existing workflow instances.

- [N/workflow Module Members](#)
- [N/workflow Module Script Sample](#)

N/workflow Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	workflow.initiate(options)	number	Server-side scripts	Initiates a workflow on-demand. This method is the programmatic equivalent of the Initiate Workflow Action action in SuiteFlow. Returns the internal ID (number) of the workflow instance used to track the workflow against the record.
	workflow.trigger(options)	number	Server-side scripts	Triggers a workflow on a record. The actions and transitions of the workflow are evaluated for the record in the workflow instance, based on the current state for the workflow instance. Returns the internal ID (number) of the workflow instance used to track the workflow against the record.

N/workflow Module Script Sample

The following example searches for a specific workflow deployed on the customer record and then executes it.

This sample script uses the `require` function so that you can copy it into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For additional information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).



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();
});

```

workflow.initiate(options)

Method Description	Initiates a workflow on-demand. This method is the programmatic equivalent of the Initiate Workflow Action action in SuiteFlow. Returns the internal ID of the workflow instance used to track the workflow against the record.
Returns	number
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types
Governance	20 usage units
Module	N/workflow Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.recordType	number	required	The record type ID of the workflow base record. For example, use 'customer', 'salesorder', or 'lead'. This is the Record Type field on the Workflow Definition Page .	2015.2
options.recordId	string number	required	The internal ID of the base record	2015.2
options.workflowId	string number	required	The internal ID (number) or script ID (string) for the workflow definition. This is the ID field on the Workflow Definition Page .	2015.2
options.defaultValues	Object	optional	The object that contains key/value pairs to set default values on fields specific to the workflow. These can include fields on the Workflow Definition Page or workflow and state Workflow Custom Fields .	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/workflow Module Script Sample](#).

```
//Add additional code
...
var workflowInstanceId = workflow.initiate({
    recordType: 'customer',
    recordId: 24,
    workflowId: 'customworkflow_myWorkFlow'
});
...
//Add additional code
```

workflow.trigger(options)

Method Description	Triggers a workflow on a record. The actions and transitions of the workflow are evaluated for the record in the workflow instance, based on the current state for the workflow instance. Returns the internal ID of the workflow instance used to track the workflow against the record.
Returns	number
Supported Script Types	All server-side scripts For more information, see SuiteScript 2.0 Script Types
Governance	20 usage units
Module	N/workflow Module
Since	2015.2

Parameters



Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.recordType	number	required	The record type ID of the workflow base record. For example, use 'customer', 'salesorder', or 'lead'. This is the Record Type field on the Workflow Definition Page .	2015.2
options.recordId	string number	required	The internal ID of the base record	2015.2
options.workflowId	string number	required	The internal ID (number) or script ID (string) for the workflow definition. This is the ID field on the Workflow Definition Page .	2015.2
options.workflowInstanceId	string number	optional	The internal ID of the workflow instance.	2015.2

Parameter	Type	Required / Optional	Description	Since
options.actionId	string number	optional	The internal ID of a button that appears on the record in the workflow. Use this parameter to trigger the workflow as if the specified button were clicked.	2015.2
options.stateId	string number	optional	The internal ID (number) or script ID (string) of the workflow instance.	2015.2

Syntax



Important: The following code snippet shows the syntax for this member. It is not a functional example. For a complete script example, see [N/workflow Module Script Sample](#).

```
//Add additional code
...
var workflowInstanceId = workflow.trigger({
    recordType: 'salesorder',
    recordId: 1234,
    workflowId: 'custworkflow_name',
    defaultValues: p
    actionId: workflowaction25
});
...
//Add additional code
```

N/xml Module

Load the xml module to validate, parse, read, and modify XML documents.

- [N/xml Module Members](#)
- [Parser Object Members](#)
- [XPath Object Members](#)
- [Node Object Members](#)
- [Document Object Members](#)
- [Element Object Members](#)
- [Attr Object Members](#)
- [N/xml Module Script Samples](#)

N/xml Module Members

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Object	xml.Parser	Object	Client and server-side scripts	Encapsulates the functionality used by NetSuite to parse XML.
	xml.XPath	Object	Client and server-side scripts	Encapsulates the functionality used by NetSuite to run XPath expressions. XPath is a standard for enumerating paths in an XML document collection.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>xml.Node</code>	Object	Client and server-side scripts	Represents a generic XML node in an XML document. A node can be a Document, Element, or Attribute.
	<code>xml.Document</code>	Object	Client and server-side scripts	Represents an entire XML document. The XML DOM presents a document as a hierarchy of node objects. Use the methods and properties available to the <code>xml.Document</code> object to manipulate the XML document and the nodes in the document tree.
	<code>xml.Element</code>	Object	Client and server-side scripts	Represents an element in an XML document. Elements may contain attributes, other elements, or text. If an element contains text, the text is represented in a text node of type <code>TEXT_NODE</code> .
	<code>xml.Attr</code>	Object	Client and server-side scripts	Represents an attribute node of an <code>xml.Element</code> object.
Method	<code>xml.escape(options)</code>	string	Client and server-side scripts	Prepares a string for use in XML by escaping XML markup, such as angle brackets, quotation marks, and ampersands.
	<code>xml.validate(options)</code>	void	Server-side scripts	Validates an XML document against an XML Schema (XSD).
Enum	<code>xml.NodeType</code>	string (read-only)	Client and server-side scripts	Enumeration that holds the string values for the supported node types. The <code>Node.nodeType</code> property is defined by one of the values in this enum.

Parser Object Members

The following members are called on the `xml.Parser` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Parser.fromString(options)</code>	<code>xml.Document</code>	Client and server-side scripts	Parses a string into a W3C XML document object.
	<code>Parser.toString(options)</code>	string	Client and server-side scripts	Converts (serializes) an <code>xml.Document</code> object into a string.

XPath Object Members

The following members are called on the `xml.XPath` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>XPath.select(options)</code>	<code>xml.Node[]</code>	Client and server-side scripts	Selects an array of nodes from an XML document using an XPath expression.

Node Object Members

The following members are called on the `xml.Node` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Node.appendChild(options)</code>	<code>xml.Node</code>	Client and server-side scripts	Appends a node after the last child node of a specific element node. Returns the new child node.
	<code>Node.cloneNode(options)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a copy of a node. Returns the copied node.
	<code>Node.compareDocumentPosition(otherNode)</code>	<code>number</code>	Client and server-side scripts	Returns a number that reflects where two nodes are located, compared to each other.
	<code>Node.hasAttributes()</code>	<code>boolean true false</code>	Client and server-side scripts	Returns <code>true</code> if the current node has child nodes or returns <code>false</code> if the current node does not have child nodes.
	<code>Node.hasChildNodes()</code>	<code>boolean true false</code>	Client and server-side scripts	Returns <code>true</code> if the current node has any attributes. Note that only element nodes can have attributes.
	<code>Node.insertBefore(options)</code>	<code>xml.Node</code>	Client and server-side scripts	Inserts a new child node before an existing child node for the current node.
	<code>Node.isDefaultNamespace(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns <code>true</code> if the specified namespace uniform resource identifier (URI) is the default namespace for the current node or returns <code>false</code> if the specified namespace is not the default namespace.
	<code>Node.isEqualNode(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns <code>true</code> if two nodes are equal or returns <code>false</code> if two nodes are not equal.
	<code>Node.isSameNode(options)</code>	<code>boolean true false</code>	Client and server-side scripts	Returns <code>true</code> if two nodes reference the same object or returns <code>false</code> if two nodes do not reference the same object.
	<code>Node.lookupNamespaceURI(options)</code>	<code>string</code>	Client and server-side scripts	Returns the namespace uniform resource identifier (URI) that matches the specified namespace prefix.
	<code>Node.lookupPrefix(options)</code>	<code>string</code>	Client and server-side scripts	Returns the namespace prefix associated with the specified namespace uniform resource identifier (URI).
	<code>Node.normalize()</code>	<code>void</code>	Client and server-side scripts	Puts all text nodes underneath a node, including attribute nodes, into a normal form.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	Node.removeChild(options)	xml.Node	Client and server-side scripts	Removes the specified child node. Returns the removed child node.
	Node.replaceChild(options)	xml.Node	Client and server-side scripts	Replaces a specific child node with another child node in a list of child nodes.
Property	Node.attributes	Object (read-only)	Client and server-side scripts	Key-value pairs for all attributes for an xml.Element node. Returns <code>null</code> for all other node types.
	Node.baseURI	string (read-only)	Client and server-side scripts	Absolute base uniform resource identifier (URI) of a node or <code>null</code> if the URI cannot be determined.
	Node.childNodes	xml.Node[] (read-only)	Client and server-side scripts	Array of all child nodes of a node or an empty array if there are no child nodes.
	Node.firstChild	xml.Node (read-only)	Client and server-side scripts	First child node for a specific node or <code>null</code> if there are no child nodes.
	Node.lastChild	xml.Node (read-only)	Client and server-side scripts	Last child node for a specific node or <code>null</code> if there is no last child node.
	Node.localName	string (read-only)	Client and server-side scripts	The local part of the qualified name of a node.
	Node.namespaceURI	string (read-only)	Client and server-side scripts	The namespace uniform resource identifier (URI) of a node or <code>null</code> if there is no namespace URI for the node.
	Node.nextSibling	xml.Node (read-only)	Client and server-side scripts	The next node in a node list or <code>null</code> if the current node is the last node.
	Node.nodeName	string (read-only)	Client and server-side scripts	Name of a node, depending on the type. For example, for a node of type xml.Element , the name is the name of the element.
	Node.nodeType	string	Client and server-side scripts	The type of node defined as a value from the xml.NodeType enum.
	Node.nodeValue	string	Client and server-side scripts	The value of a node, depending on its type.
	Node.ownerDocument	xml.Document (read-only)	Client and server-side scripts	The root element for a node as a xml.Document object.
	Node.parentNode	xml.Node (read-only)	Client and server-side scripts	The parent node of a node.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>Node.prefix</code>	string	Client and server-side scripts	The namespace prefix of the node, or <code>null</code> if the node does not have a namespace.
	<code>Node.previousSibling</code>	<code>xml.Node</code> (read-only)	Client and server-side scripts	The previous node in a node list or <code>null</code> if the current node is the first node.
	<code>Node.textContent</code>	string	Client and server-side scripts	The textual content of a node and its descendants.

Document Object Members

Note: In addition to the Document object members, Document objects inherit the members of the Node object. The methods and properties associated with a Node object can be used as members of a Document object. For more information, see [Node Object Members](#).

The following members are called on the `xml.Document` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	<code>Document.adoptNode(options)</code>	<code>xml.Node</code>	Client and server-side scripts	Attempts to adopt a node from another document to this document.
	<code>Document.createAttribute(options)</code>	<code>xml.Attr</code>	Client and server-side scripts	Creates an attribute node of type <code>ATTRIBUTE_NODE</code> with the optional specified value.
	<code>Document.createAttributeNS(namespace, name, options)</code>	<code>xml.Attr</code>	Client and server-side scripts	Creates an attribute node of type <code>ATTRIBUTE_NODE</code> , with the specified namespace value and optional specified value.
	<code>Document.createCDATASection(data, options)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a CDATA section node of type <code>DOCUMENT_FRAGMENT_NODE</code> with the specified data.
	<code>Document.createComment(comment, options)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a Comment node of type <code>COMMENT_NODE</code> with the specified string.
	<code>Document.createDocumentFragment(contents)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a node of type <code>DOCUMENT_FRAGMENT_NODE</code> .
	<code>Document.createElement(name, options)</code>	<code>xml.Element</code>	Client and server-side scripts	Creates a new node of type <code>ELEMENT_NODE</code> with the specified name.
	<code>Document.createElementNS(namespace, name, options)</code>	<code>xml.Element</code>	Client and server-side scripts	Creates a new node of type <code>ELEMENT_NODE</code> with the specified namespace URI and name.
	<code>Document.createProcessingInstruction(target, data, options)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a new node of type <code>PROCESSING_INSTRUCTION_NODE</code> with the specified target and data.
	<code>Document.createTextNode(data, options)</code>	<code>xml.Node</code>	Client and server-side scripts	Creates a new node of type <code>TEXT_NODE</code> .

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
	<code>Document.getElementById(options)</code>	<code>xml.Element</code>	Client and server-side scripts	Returns the element that has an ID attribute with the specified value as an <code>xml.Element</code> object.
	<code>Document.getElementsByTagName(tagName)</code>	<code>xml.Element[]</code>	Client and server-side scripts	Returns an array of <code>xml.Element</code> objects with a specific tag name, in the order in which they appear in the XML document.
	<code>Document.getElementsByTagNameNS(namespace, tagName)</code>	<code>xml.Element[]</code>	Client and server-side scripts	Returns an array of <code>xml.Element</code> objects with a specific tag name and namespace, in the order in which they appear in the XML document.
	<code>Document.importNode(options)</code>	<code>xml.Node</code>	Client and server-side scripts	Imports a node from another document to this document. Creates a new copy of the source node.
Property	<code>Document.doctype</code>	Object (read-only)	Client and server-side scripts	Returns a node of type <code>DOCUMENT_TYPE_NODE</code> that represents the doctype of the XML document.
	<code>Document.documentElement</code>	<code>xml.Element</code> (read-only)	Client and server-side scripts	Root node of the XML document.
	<code>Document.documentURI</code>	string (read-only)	Client and server-side scripts	Location of the document or <code>null</code> if undefined.
	<code>Document.inputEncoding</code>	string (read-only)	Client and server-side scripts	Encoding used for an XML document at the time the document was parsed.
	<code>Document.xmlEncoding</code>	string (read-only)	Client and server-side scripts	Part of the XML declaration, the XML encoding of the XML document.
	<code>Document.xmlStandalone</code>	boolean <code>true</code> <code>false</code>	Client and server-side scripts	Part of the XML declaration, returns <code>true</code> if the current XML document is standalone or returns <code>false</code> if it is not.
	<code>Document.xmlVersion</code>	string	Client and server-side scripts	Part of the XML declaration, the version number of the XML document.

Element Object Members

Note: In addition to the Element object members, Element objects inherit the members of the Node object. The methods and properties associated with a Node object can be used as members of a Element object. For more information, see [Node Object Members](#).

The following members are called on the `xml.Element` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Method	Element.getAttribute(options)	string	Client and server-side scripts	Returns the value of the specified attribute.
	Element.getAttributeNode(options)	xml.Attr	Client and server-side scripts	Retrieves an attribute node by name.
	Element.getAttributeNodeNS(options)	string	Client and server-side scripts	Returns an attribute node with the specified namespace URI and local name.
	Element.getAttributeNS(options)	xml.Attr	Client and server-side scripts	Returns an attribute value with the specified namespace URI and local name.
	Element.getElementsByTagName(options)	xml.Element[]	Client and server-side scripts	Returns an array of descendant xml.Element objects with a specific tag name, in the order in which they appear in the XML document.
	Element.getElementsByTagNameNS(options)	xml.Element[]	Client and server-side scripts	Returns an array of descendant xml.Element objects with a specific tag name and namespace, in the order in which they appear in the XML document.
	Element.hasAttribute(options)	boolean true false	Client and server-side scripts	Returns true if the current element has an attribute with the specified name or if that attribute has a default value. Otherwise, returns false .
	Element.hasAttributeNS(options)	boolean true false	Client and server-side scripts	Returns true if the current element has an attribute with the specified local name and namespace or if that attribute has a default value. Otherwise, returns false .
	Element.removeAttribute(options)	void	Client and server-side scripts	Removes the attribute with the specified name.
	Element.removeAttributeNode(options)	xml.Attr	Client and server-side scripts	Removes the attribute specified as a xml.Attr object.
	Element.removeAttributeNS(options)	void	Client and server-side scripts	Removes the attribute with the specified namespace URI and local name.
	Element.setAttribute(options)	void	Client and server-side scripts	Adds a new attribute with the specified name. If an attribute with that name is already present in the element, its value is changed to the value specified in method argument.
	Element.setAttributeNode(options)	xml.Attr	Client and server-side scripts	Adds the specified attribute node. If an attribute with the same name is already present

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
				in the element, it is replaced by the new one.
	<code>Element.setAttributeNodeNS(options)</code>	<code>xml.Attr</code>	Client and server-side scripts	Adds the specified attribute node. If an attribute with the same local name and namespace URI is already present in the element, it is replaced by the new one.
	<code>Element.setAttributeNS(options)</code>	void	Client and server-side scripts	Adds a new attribute with the specified name and namespace URI. If an attribute with the same name and namespace URI is already present in the element, its value is changed to the value specified in method argument.
Property	<code>Element.tagName</code>	string (read-only)	Client and server-side scripts	The tag name of this <code>xml.Element</code> object.

Attr Object Members

The following members are called on the `xml.Attr` object.

Member Type	Name	Return Type / Value Type	Supported Script Types	Description
Property	<code>Attr.name</code>	string (read-only)	Client and server-side scripts	The name of an attribute.
	<code>Attr.ownerElement</code>	<code>xml.Element</code> (read-only)	Client and server-side scripts	The <code>xml.Element</code> object that is the parent of the <code>xml.Attr</code> object.
	<code>Attr.specified</code>	boolean <code>true</code> <code>false</code>	Client and server-side scripts	Returns <code>true</code> if the attribute value is set in the parsed XML document, and <code>false</code> if it is a default value in a DTD or Schema.
	<code>Attr.value</code>	string	Client and server-side scripts	Value of an attribute. The value of the attribute is returned as a string. Character and general entity references are replaced with their values.

N/xml Module Script Samples

These samples use the `require` function, so that you can copy each script into the debugger and test it. Keep in mind that you must use the `define` function in your entry point script (the script you attach to a script record). For more information, see [SuiteScript 2.0 Script Basics](#) and [SuiteScript 2.0 Script Types](#).

For help with writing scripts in SuiteScript 2.0, see [SuiteScript 2.0 Hello World](#) and [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

The N/xml module sample references the following XML file, BookSample.xml:

```
<bookstore xmlns:b="http://www.qualifiednamespace.com/book">
  <b:book category="cooking">
```

```

    <b:title lang="en">Everyday Italian</b:title>
    <b:author>Giada De Laurentiis</b:author>
    <b:year>2005</b:year>
    <b:price>30.00</b:price>
  </b:book>
  <b:book category="children">
    <b:title lang="en">Harry Potter</b:title>
    <b:author>J K. Rowling</b:author>
    <b:year>2005</b:year>
    <b:price>29.99</b:price>
  </b:book>
  <b:book category="web">
    <b:title lang="en">XQuery Kick Start</b:title>
    <b:author>James McGovern</b:author>
    <b:author>Per Bothner</b:author>
    <b:author>Kurt Cagle</b:author>
    <b:author>James Linn</b:author>
    <b:author>Vaidyanathan Nagaraajan</b:author>
    <b:year>2003</b:year>
    <b:price>49.99</b:price>
  </b:book>
  <b:book category="web" cover="paperback">
    <b:title lang="en">Learning XML</b:title>
    <b:author>Erik T. Ray</b:author>
    <b:year>2003</b:year>
    <b:price>39.95</b:price>
  </b:book>
</bookstore>

```

The following Suitelet example loads an XML file from the file cabinet, iterates through the individual book nodes, and accesses the child node values using two common methods: (through firstChild/nextSibling/etc and through getElementsByTagName)

```

/**
 * @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 : '//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);
}
};
});

```

The following output is produced from the sample code when used with the BookSample.xml document:

```

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.

```

In the following example, the XML parser parses the XML string stored in the `xmlString` variable. Then, the script selects all `config` elements in the `xmlDocument` node, loops through them and logs their content.

```

/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
require(['N/xml'], function(xml) {
    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'
    });

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

```

xml.Parser

Object Description	Encapsulates the functionality used by NetSuite to parse an XML document. For a complete list of this object's methods, see Parser Object Members .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var parserObj = xml.Parser;
...
//Add additional code
```

Parser.fromString(options)

Method Description	<p>Parses a String into a W3C XML document object. This API is useful if you want to navigate/query a structured XML document more effectively using either the Document API or NetSuite built-in XPath functions.</p> <div>  Note: You can also use this method to validate your XML. If you pass a malformed string in as the <code>options.text</code> argument, <code>Parser.fromString</code> returns an <code>SSS_XML_DOM_EXCEPTION</code> error. </div>
Returns	<code>xml.Document</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.text</code>	string	Required	String being converted to an <code>xml.Document</code> .

Errors

Error Code	Thrown If
<code>SSS_XML_DOM_EXCEPTION</code>	The input XML string is malformed.

Syntax

```
//Add additional code
...
var xmlDocument = xml.Parser.fromString({
    text : xmlStringContent
});
...
//Add additional code
```

Parser.toString(options)

Method Description	Converts (serializes) an xml.Document object into a string. This API is useful, for example, if you want to serialize and store an xml.Document in a custom field.
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.document	xml.Document	Required	XML document being serialized.

Syntax

```
//Add additional code
...
var xmlStringContent = xml.Parser.toString({
    document : xmlDocument
});
...
//Add additional code
```

xml.XPath

Object Description	Encapsulates the functionality to run XPath expressions. For a complete list of this object's methods, see XPath Object Members .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var xpath = xml.XPath;
...
```

```
//Add additional code
```

XPath.select(options)

Method Description	Selects an array of nodes from an XML that match an XPath expression.
Returns	xml.Node[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.node	xml.Node	Required	XML node being queried.
options.xpath	string	Required	XPath expression used to query node.

Syntax

```
//Add additional code
...
var bookNode = xml.XPath.select({
  node : xmlDocument,
  xpath : '//book'
});
...
//Add additional code
```

xml.Node

Object Description	Represents a single node in an XML document tree. The XML DOM presents a document as a hierarchy of node objects. See the xml.NodeType enum for a list of possible node types. You can use this object to work with a child node, or nested nodes. NetSuite supports a subset of W3C DOM methods. For a complete list of this object's methods and properties, see Node Object Members . For other code snippets that use this object, see the syntax sample that follows, as well as Node.childNodes and N/xml Module Script Samples .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var bookNode = xml.XPath.select({
  node : xmlDocument,
  xpath : '//book'
});
...
//Add additional code
```

Node.appendChild(options)

Method Description	Appends a node after the last child node of a specific element node. Returns the new child node.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.newChild	xml.Node	Required	xml.Node object to append.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.	Node cannot be appended.

Syntax

```
//Add additional code
...
var bookShelf = xml.Parser.fromString(file.load('SuiteScripts/books.xml').getContents());

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



```

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

var newbook = bookShelf.appendChild({
    newChild : newBookNode
});
...
//Add additional code

```

Node.cloneNode(options)

Method Description	Creates a copy of a node. Returns the copied node.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.deep	boolean <code>true</code> <code>false</code>	Optional	Use <code>true</code> to clone the node, attributes, and all descendents. Use <code>false</code> to only clone the node and attributes.

Syntax

```

//Add additional code
...
var copiednode = elem[0].cloneNode({
    deep : true
});
...
//Add additional code

```

Node.compareDocumentPosition(options)

Method Description	<p>Returns a number that reflects where two nodes are located, compared to each other. Returns one of the following numbers:</p> <ul style="list-style-type: none"> 1. The two nodes do not belong to the same document.
---------------------------	---

	<ul style="list-style-type: none"> 2. The specified node comes before the current node. 4. The specified node comes after the current node. 8. The specified node contains the current node. 16. The current node contains the specified node. 32. The specified and current nodes do not have a common container node or the two nodes are different attributes of the same node. <div>  Note: The return value can be a combination of the above values. For example, a return value of 20 means the specified node is contained by the current node, a value of 16, and the specified node follows the current node, a value of 4. </div> <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	number
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.other	xml.Node	Required	The node to compare with the current node.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected xml.Node or subclass: other	The options.other is of type xml.Node .

Syntax

```
//Add additional code
...
var posCode = elem[0].compareDocumentPosition({
  other : parentNode[0]
});
...
//Add additional code
```

Node.hasAttributes()

Method Description	Returns <code>true</code> if the current node has attributes defined, or <code>false</code> otherwise.
---------------------------	--

	 Important: This method is not supported on Internet Explorer.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var hasAttributes = parentNode[0].hasAttributes()
...
//Add additional code
```

Node.hasChildNodes()

Method Description	Returns <code>true</code> if the current node has child nodes or returns <code>false</code> if the current node does not have child nodes.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var hasChildren = parentNode[0].hasChildNodes()
...
//Add additional code
```

Node.insertBefore(options)

Method Description	Inserts a new child node before an existing child node for the current node. If the new child node is already in the list of children, this method removes the new child node and inserts it again.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.newChild	xml.Node	Required	The new child node to insert.
options.refChild	xml.Node	Required	The node before which to insert the new child node. If <code>refChild</code> is, the method inserts the new node at the end of the list of children.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.	Node cannot be inserted.

Syntax

```
//Add additional code
...
var insertednode = parentNode[0].insertBefore({
    newChild : elemList1[0],
    refChild : elemList2[0]
});
...
//Add additional code
```

Node.isDefaultNamespace(options)

Method Description	Returns <code>true</code> if the specified namespace uniform resource identifier (URI) is the default namespace for the current node or returns <code>false</code> if the specified namespace is not the default namespace. See also Node.namespaceURI .
	 Important: This method is not supported on Internet Explorer.
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	The namespace URI to compare.

Syntax

```
//Add additional code
...
var isDefault = parentNode[0].isDefaultNamespace({
    namespaceURI : '*'
});
...
//Add additional code
```

Node.isEqualNode(options)

Method Description	<p>Returns <code>true</code> if two nodes are equal or returns <code>false</code> if two nodes are not equal. The two nodes are equal if they meet the following conditions:</p> <ul style="list-style-type: none"> Both nodes have the same type. Both nodes have the same attributes and attribute values. The order of the attributes is not considered. Both nodes have equal lists of child nodes and the child nodes appear in the same order. <div>  Note: Two nodes may be equal, even if they are not the same. See <code>Node.isSameNode(options)</code>. </div> <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.other	<code>xml.Node</code>	Required	The node to compare with the current node.

Syntax

```
//Add additional code
```

```

...
var isEqual = elem[0].isEqualNode({
    other : node
});
...
//Add additional code

```

Node.isSameNode(options)

Method Description	<p>Returns <code>true</code> if two nodes reference the same object or returns <code>false</code> if two nodes do not reference the same object. If two nodes are the same, all attributes have the same values and you can use methods on the two nodes interchangeably.</p> <div>  Note: Two nodes that are the same are also equal. See Node.isEqualNode(options). </div> <div>  Important: This method is not supported on Internet Explorer or Firefox. </div>
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.other	xml.Node	Required	The node to compare with the current node.

Syntax

```

//Add additional code
...
var isSame = elem[0].isSameNode({
    other : node
});
...
//Add additional code

```

Node.lookupNamespaceURI(options)

Method Description	Returns the namespace uniform resource identifier (URI) that matches the specified namespace prefix.
---------------------------	--

	Returns <code>null</code> if the specified prefix does not have an associated URI.
	 Important: This method is not supported on Internet Explorer.
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.prefix	string	Required	Namespace prefix associated with the namespace URI.

Syntax

```
//Add additional code
...
var uri = parentNode[0].lookupNamespaceURI({
  prefix : '*'
});
...
//Add additional code
```

Node.lookupPrefix(options)

Method Description	<p>Returns the namespace prefix associated with the specified namespace uniform resource identifier (URI). Returns <code>null</code> if the specified URI does not have an associated prefix. If more than one prefix is associated with the namespace prefix, the namespace returned by this method depends on the module implementation.</p>  Important: This method is not supported on Internet Explorer.
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI associated the namespace prefix.

Syntax

```
//Add additional code
...
var prefix = parentNode[0].lookupPrefix({
    namespaceURI : ''
});
...
//Add additional code
```

Node.normalize()

Method Description	Puts all text nodes underneath a node, including attribute nodes, into a normal form. In normal form, only structure (such as elements, comments, processing instructions, CDATA sections, and entity references) separates text nodes. After normalization, there are no adjacent or empty text nodes. Use this method if you require a particular document tree structure and want to make sure that the XML DOM view of a document is identical when you save and reload it.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
node.normalize();
...
//Add additional code
```

Node.removeChild(options)

Method Description	Removes the specified child node.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None

Module	N/xml Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.oldChild	xml.Node	Required	Node to remove.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NOT_FOUND_ERR: An attempt is made to reference a node in a context where it does not exist.	Node cannot be removed.

Syntax

```
//Add additional code
...
var removednode = parentNode[0].removeChild({
  oldChild : node
});
...
//Add additional code
```

Node.replaceChild(options)

Method Description	Replaces a specific child node with another child node in a list of child nodes. If the new child node to add already exists in the list of child nodes, the node is first removed.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.newChild	xml.Node	Required	New child node to add.
options.oldChild	xml.Node	Required	Child node to be replaced with the new node.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NOT_FOUND_ERR: An attempt is made to reference a node in a context where it does not exist.	Child node cannot be found.
SSS_XML_DOM_EXCEPTION	HIERARCHY_REQUEST_ERR: An attempt was made to insert a node where it is not permitted.	Child node cannot be replaced.

Syntax

```
//Add additional code
...
var replacednode = parentNode.replaceChild({
  newChild : elem[2],
  oldChild : elem[1]
});
...
//Add additional code
```

Node.attributes

Property Description	Key-value pairs for all attributes for an <code>xml.Element</code> node. Returns <code>null</code> for all other node types.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var attribs = elem[0].attributes;
...
//Add additional code
```

Node.baseURI

Property Description	<p>Absolute base uniform resource identifier (URI) of a node or <code>null</code> if the URI cannot be determined. For client scripts, this property always returns <code>null</code>.</p> <div>  Note: The format of this value is browser-specific. </div>
Type	string (read-only)
Supported Script Types	All script types

	For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var baseuri = parentNode[0].baseURI;
...
//Add additional code
```

Node.childNodes

Property Description	Array of all child nodes of a node or an empty array if there are no child nodes.
Type	xml.Node[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var childnodes = parentNode[0].childNodes;
...
//Add additional code
```

Node.firstChild

Property Description	The first child node of a node, or <code>null</code> if there are no child nodes.
Type	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeValue1 = bookNode[0].firstChild.nextSibling.textContent;
...
```

```
//Add additional code
```

Node.lastChild

Property Description	The last child node of a node, or <code>null</code> if there are no child nodes.
Type	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeValue = parentNode[0].lastChild.previousSibling.textContent;
...
//Add additional code
```

Node.localName

Property Description	The local part of the qualified name of a node.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var localname = parentNode[0].localName;
...
//Add additional code
```

Node.namespaceURI

Property Description	The namespace uniform resource identifier (URI) of a node or <code>null</code> if there is no namespace URI for the node.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module

Since	2015.2
-------	--------

Syntax

```
//Add additional code
...
var uri = parentNode[0].namespaceURI;
...
//Add additional code
```

Node.nextSibling

Property Description	The next node in a node list or <code>null</code> if the current node is the last node.
Type	<code>xml.Node</code> (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeName = parentNode[0].firstChild.nextSibling.textContent;
...
//Add additional code
```

Node.nodeName

Property Description	<p>Name of a node, depending on the type. For example, for a node of type <code>xml.Element</code>, the name is the name of the element.</p> <div>  Note: On Chrome, this property also includes the namespace or prefix. </div>
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeName = parentNode[0].firstChild.nodeName;
...
//Add additional code
```

Node.nodeType

Property Description	The type of node as an enum. For all possible values of this property, see xml.NodeType .
Type	xml.NodeType
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeType = parentNode[0].firstChild.nodeType;
...
//Add additional code
```

Node.nodeValue

Property Description	The value of a node, depending on its type. If the value is <code>null</code> , setting this value has no effect.
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var nodeValue = parentNode[0].firstChild.nodeValue;
...
//Add additional code
```

Node.ownerDocument

Property Description	The root element for a node as a xml.Document object. Use this object to create new nodes with Document.createElement(options) or Document.createElementNS(options) .
Type	xml.Document
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var doc = parentNode[0].ownerDocument;
...
//Add additional code
```

Node.parentNode

Property Description	The parent node of a node. All node types, except xml.Attr , xml.Document , DocumentFragment , Entity , and Notation can have a parent node. See xml.NodeType for possible node types.
Type	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code ...
var nodevalue = parentNode[0].lastChild.parentNode.textContent;
...
//Add additional code
```

Node.prefix

Property Description	The namespace prefix of the node, or <code>null</code> if the node does not have a namespace. If the value is <code>null</code> , setting it has no effect, including read-only node types.
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NAMESPACE_ERR: An attempt is made to create or change an object in a way which is incorrect with regard to namespaces.	Cannot edit the node prefix.

Syntax

```
//Add additional code
```

```

...
var namespacePrefix = parentNode[0].firstChild.prefix;
...
//Add additional code

```

Node.previousSibling

Property Description	The previous node in a node list or <code>null</code> if the current node is the first node.
Type	<code>xml.Node</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```

//Add additional code
...
var nodeValue = parentNode[0].lastChild.previousSibling.textContent;
...
//Add additional code

```

Node.textContent

Property Description	The textual content of a node and its descendants. If the value is <code>null</code> , then setting it has no effect. If you set this value, any child nodes are removed and replaced by a single text node with this string as a value.
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```

//Add additional code
...
var nodeValue = parentNode[0].firstChild.textContent;
...
//Add additional code

```

xml.Document

Object Description	Represents an entire XML document. The XML DOM presents a document as a hierarchy of node objects. Use the methods and properties available to the <code>xml.Document</code> object to manipulate the XML document and the nodes in the document tree. For a list of this object's methods and properties, see Document Object Members .
---------------------------	---

	An XML document object is also a node of type DOCUMENT_NODE. In addition to the Document object members, Document objects inherit the members of the Node object. For a complete list of these methods and properties, see Node Object Members .
Supported Script Types	All script types
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var xmlDocument = xml.Parser.fromString({
    text : xmlFileContent
});
...
//Add additional code
```

Document.adoptNode(options)

Method Description	Attempts to adopt a node from another document to this document. If successful, this method changes the Node.ownerDocument property of the source node, its children, and any attribute nodes to the current document. If the source node has a parent node, the parent node is first removed from the child list of its own parent node. <div> Important: This method is not supported on Internet Explorer.</div>
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.source	xml.Node	Required	Source node to add as a child into the current node object.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NOT_FOUND_ERR: An attempt is made to reference a node in a context where it does not exist.	Node cannot be adopted.

Syntax

```
//Add additional code
...
var adoptedNode = xmlDocument1.adoptNode({
    source : sourceNode,
});
...
//Add additional code
```

Document.createAttribute(options)

Method Description	Creates an attribute node of type <code>ATTRIBUTE_NODE</code> with the optional specified value and returns the new <code>xml.Attr</code> object. The <code>localName</code> , <code>prefix</code> , and <code>namespaceURI</code> properties of the new node are set to <code>null</code> .
Returns	<code>xml.Attr</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The `options` parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.name</code>	string	Required	Name of the new attribute node.
<code>options.value</code>	string	Optional	Value for the attribute node. If unspecified, the value is an empty string.

Errors

Error Code	Message	Thrown If
<code>SSS_XML_DOM_EXCEPTION</code>	<code>INVALID_CHARACTER_ERR</code> : An invalid or illegal XML character is specified.	Attribute with the specified name or value cannot be created.

Syntax

```
//Add additional code
...
var attr = xmlDocument.createAttribute({
    name : 'lang',
    value : 'fr'
});
...
```

```
//Add additional code
```

Document.createAttributeNS(options)

Method Description	<p>Creates an attribute node of type <code>ATTRIBUTE_NODE</code>, with the specified namespace value and optional specified value, and returns the new <code>xml.Attr</code> object. The <code>Node.localName</code>, <code>Node.prefix</code>, and <code>Node.namespaceURI</code> properties of the new node are set to <code>null</code>.</p> <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	<code>xml.Attr</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
<code>options.namespaceURI</code>	string	Required	Namespace URI of the attribute to create. Value can be <code>null</code> .
<code>options.qualifiedName</code>	string	Required	Qualified name of the new attribute node.
<code>options.value</code>	string	Optional	Value for the attribute node. If unspecified, the value is an empty string.

Errors

Error Code	Message	Thrown If
<code>SSS_XML_DOM_EXCEPTION</code>	<code>INVALID_CHARACTER_ERR</code> : An invalid or illegal XML character is specified.	Attribute with the specified value cannot be created.

Syntax

```
//Add additional code
...
var attr = xmlDocument.createAttributeNS({
  namespaceURI : '*',
  qualifiedName : 'lang',
  value : 'fr'
});
...
//Add additional code
```

Document.createCDATASection(options)

Method Description	Creates a CDATA section node of type DOCUMENT_FRAGMENT_NODE with the specified data and returns the new xml.Node object.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 Note: The options parameter is a JavaScript object.
--

Parameter	Type	Required / Optional	Description
options.data	string	Required	Data for the new CDATA section node.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: data	Cannot create CDATA section node with the specified data.

Syntax

<pre>//Add additional code ... var newNode = xmlDocument.createCDATASection({ data : 'Limited Edition.' }); ... //Add additional code</pre>

Document.createComment(options)

Method Description	Creates a Comment node of type COMMENT_NODE with the specified string.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.data	string	Required	Data for the Comment node.

Syntax

```
//Add additional code
...
var newNode = xmlDocument.createComment({
    data : 'This is a comment.'
});
...
//Add additional code
```

Document.createDocumentFragment()

Method Description	Creates a node of type DOCUMENT_FRAGMENT_NODE and returns the new xml.Node object.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var newNode = xmlDocument.createDocumentFragment();
...
//Add additional code
```

Document.createElement(options)

Method Description	Creates a new node of type ELEMENT_NODE with the specified name and returns the new xml.Element node. The Node.localName , Node.prefix , and Node.namespaceURI properties of the new node are set to null.
Returns	xml.Element
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module

Since	2015.2
-------	--------

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.tagName	string	Required	Name of the element to create.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INVALID_CHARACTER_ERR: An invalid or illegal XML character is specified.	Element cannot be created with the specified tagName value.

Syntax

```
//Add additional code
...
var elem = xmlDocument.createElement({
  tagName : 'book'
});
...
//Add additional code
```

Document.createElementNS(options)

Method Description	Creates a new node of type ELEMENT_NODE with the specified namespace URI and name and returns the new xml.Element object. The Node.localName , Node.prefix , and Node.namespaceURI properties of the new node are set to <code>null</code> .
Returns	xml.Element
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the element to create. Can be <code>null</code> .
options.qualifiedName	string	Required	Qualified name of the element to create.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INVALID_CHARACTER_ERR: An invalid or illegal XML character is specified.	Element with the specified namespace cannot be created.

Syntax

```
//Add additional code
...
var elem = xmlDocument.createElementNS({
  namespaceURI : '*',
  qualifiedName : 'book'
});
...
//Add additional code
```

Document.createProcessingInstruction(options)

Method Description	Creates a new node of type PROCESSING_INSTRUCTION_NODE with the specified target and data and returns the new xml.Node object. The following example shows a sample processing instruction: <code><?xml version="1.0"?></code> Use a processing instruction node to keep processor-specific information in the text of the XML document.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.target	string	Required	Target part of the processing instruction.
options.data	string	Required	Data for the processing instruction.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INVALID_CHARACTER_ERR: An invalid or illegal XML character is specified.	Processing instruction node cannot be created with the specified target or data.

Syntax

```
//Add additional code
...
var newNode = xmlDocument.createProcessingInstruction({
    target : 'xml'
    data : 'version="1.0"'
});
...
//Add additional code
```

Document.createTextNode(options)

Method Description	Creates a new text node and returns the new xml.Node object.
Returns	xml.Node
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.data	string	Required	Data for the text node.

Syntax

```
//Add additional code
...
var newNode = xmlDocument.createTextNode({
    data : 'Sample Title'
});
...
//Add additional code
```

Document.getElementById(options)

Method Description	Returns the element that has an ID attribute with the specified value as an xml.Element object. Returns <code>null</code> if no such element exists.
Returns	xml.Element
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .

Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.elementId	string	Required	Unique ID value for an element.

Syntax

```
//Add additional code
...
var elem = xmlDocument.getElementById({
  elementId : 'id12345'
});
...
//Add additional code
```

Document.getElementsByTagName(options)

Method Description	Returns an array of xml.Element objects with a specific tag name, in the order in which they appear in the XML document.
Returns	xml.Element[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.tagName	string	Required	Case-sensitive tag name of the element to match on. Use the * wildcard to match all elements.

Syntax

```
//Add additional code
...
```

```

var elem = xmlDocument.getElementsByTagName({
    tagName : 'book'
});
...
//Add additional code

```

Document.getElementsByTagNameNS(options)

Method Description	<p>Returns an array of xml.Element objects with a specific tag name and namespace, in the order in which they appear in the XML document.</p> <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	xml.Element []
Supported Script Types	<p>All script types</p> <p>For more information, see SuiteScript 2.0 Script Types.</p>
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI to match on. Use the * wildcard to match all namespaces.
options.localName	string	Required	Localname property to match on. Use the * wildcard to match all local names.

Syntax

```

//Add additional code
...
var elem = xmlDocument.getElementsByTagNameNS({
    namespaceURI : '*',
    localName : 'book'
});
...
//Add additional code

```

Document.importNode(options)

Method Description	<p>Imports a node from another document to this document. This method creates a new copy of the source node.</p> <p>If the <code>deep</code> parameter is set to <code>true</code>, it imports all children of the specified node. If set to <code>false</code>, it imports only the node itself.</p>
---------------------------	---

	Method returns the imported <code>xml.Node</code> object.
Returns	<code>xml.Node</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.importedNode	<code>xml.Node</code>	Required	Node from another XML document to import.
options.deep	boolean <code>true</code> <code>false</code>	Required	Use <code>true</code> to import the node, its attributes, and all descendents. Use <code>false</code> to only import the node and its attributes.  Important: This parameter is not supported on Internet Explorer.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NOT_SUPPORTED_ERR: The implementation does not support the requested type of object or operation.	Node cannot be imported.

Syntax

```
...
var importedNode = xmlDocument1.importNode({
    importedNode : foreignNode,
    deep : true
});
...
```

Document.doctype

Property Description	The doctype of the XML document.  Important: This property is not supported on Internet Explorer.
Type	<code>xml.Element</code> (read-only)
Supported Script Types	All script types

	For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var doctype = xmlDocument.doctype;
...
//Add additional code
```

Document.documentElement

Property Description	Root node of the XML document. Use this property to directly access the xml.Element object that represents the root node of an XML document.
Type	xml.Element (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var root = xmlDocument.documentElement;
...
//Add additional code
```

Document.documentURI

Property Description	Location of the document or <code>null</code> if undefined.
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var documentURI = xmlDocument.documentURI;
```

```
...
//Add additional code
```

Document.inputEncoding

Property Description	<p>Encoding used for an XML document at the time the document was parsed. When parsing an XML document with the following declaration, the inputEncoding property is UTF-8:</p> <pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?></pre> <div>  Important: The value of this property is browser-specific. </div>
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var encoding = xmlDocument.inputEncoding;
...
//Add additional code
```

Document.xmlEncoding

Property Description	<p>Part of the XML declaration, the XML encoding of the XML document. In the following declaration, the xmlEncoding property is UTF-8:</p> <pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?></pre> <div>  Important: This property is not supported on Internet Explorer or Firefox. </div>
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var encoding = xmlDocument.xmlEncoding;
...
//Add additional code
```

Document.xmlStandalone

Property Description	<p>Part of the XML declaration, returns <code>true</code> if the current XML document is standalone or returns <code>false</code> if it is not.</p> <p>In the following declaration, the <code>xmlStandalone</code> property is <code>true</code>:</p> <pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?></pre> <div>  Important: This property is not supported on Internet Explorer or Firefox. </div>
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var isStandalone = xmlDocument.xmlStandalone;
...
//Add additional code
```

Document.xmlVersion

Property Description	<p>Part of the XML declaration, the version number of the XML document.</p> <p>In the following declaration, the <code>xmlVersion</code> property is 1.0:</p> <pre><?xml version="1.0" encoding="UTF-8" standalone="yes"?></pre> <div>  Important: This property is not supported on Internet Explorer or Firefox. </div>
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	The implementation does not support the requested type of object or operation.	Cannot edit the XML version for the document.

Syntax

```
//Add additional code
```

```
...
var version = xmlDocument.xmlVersion;
...
//Add additional code
```

xml.Element

Object Description	Represents an element in an XML document. Elements may contain attributes, other elements, or text. If an element contains text, the text is represented in a text node of type TEXT_NODE. For example, the following element year contains a text node with the value of 2015: <pre><year>2015</year></pre> For a list of this object's methods and properties, see Element Object Members . An XML element object is also a node of type ELEMENT_NODE. In addition to the Element object members, Element objects inherit the members of the Node object. For a complete list of these methods and properties, see Node Object Members .
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var elem = parentNode[0].getElementsByTagName({tagName : 'title'});
...
//Add additional code
```

Element.getAttribute(options)

Method Description	Returns the value of the specified attribute.
Returns	xml.Attr
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	Name of the attribute for which to return the value.

Syntax

```
//Add additional code
...
var attr = elem[0].getAttribute({
  name : 'lang'
});
...
//Add additional code
```

Element.getAttributeNode(options)

Method Description	Retrieves an attribute node by name.  Important: This method is not supported on Internet Explorer.
Returns	xml.Attr
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	The name of the attribute to return.

Syntax

```
//Add additional code
...
var attr = elem[0].getAttributeNode({
  name : 'lang'
});
...
//Add additional code
```

Element.getAttributeNodeNS(options)

Method Description	Returns an attribute node with the specified namespace URI and local name.  Important: This method is not supported on Internet Explorer.
Returns	string

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the attribute to return. Value can be <code>null</code> .
options.localName	string	Required	Local name of the attribute to return.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: namespaceURI	Attribute node with the specified namespace cannot be retrieved.

Syntax

```
//Add additional code
...
var attr = elem[0].getAttributeNodeNS({
  namespaceURI : '*'
  localName : 'lang'
});
...
//Add additional code
```

Element.getAttributeNS(options)

Method Description	Returns an attribute value with the specified namespace URI and local name. <div> Important: This method is not supported on Internet Explorer.</div>
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the attribute to return. Value can be <code>null</code> .
options.localName	string	Required	Local name of the attribute to return.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: namespaceURI	Attribute with the specified namespace cannot be retrieved.

Syntax

```
//Add additional code
...
var attr = elem[0].getAttributeNS({
  namespaceURI : '*'
  localName : 'lang'
});
...
//Add additional code
```

Element.getElementsByTagName(options)

Method Description	Returns an array of descendant xml.Element objects with a specific tag name, in the order in which they appear in the XML document.
Returns	xml.Element[]
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.tagName	string	Required	Case-sensitive tag name of the element to match on. Use the * wildcard to match all elements.

Syntax

```
//Add additional code
...
var elem = parentNode[0].getElementsByTagName({tagName : 'title'});
...
//Add additional code
```

Element.getElementsByTagNameNS(options)

Method Description	Returns an array of descendant xml.Element objects with a specific tag name and namespace, in the order in which they appear in the XML document. <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	xml.Element []
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI to match on. Use the * wildcard to match all namespaces.
options.localName	string	Required	Localname property to match on. Use the * wildcard to match all local names.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: namespaceURI	Elements with the specified namespace cannot be retrieved.

Syntax

```
//Add additional code
...
var elem = parentNode[0].getElementsByTagNameNS({
  namespaceURI : '*',
  localName : 'lang'
});
...
```

```
//Add additional code
```

Element.hasAttribute(options)

Method Description	Returns <code>true</code> if the current element has an attribute with the specified name or if that attribute has a default value. Otherwise, returns <code>false</code> . <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	Name of the attribute to match on.

Syntax

```
//Add additional code
...
var attrExists = elem[0].hasAttribute({
  name : 'lang'
});
...
//Add additional code
```

Element.hasAttributeNS(options)

Method Description	Returns <code>true</code> if the current element has an attribute with the specified local name and namespace or if that attribute has a default value. Otherwise, returns <code>false</code> . <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module

Since	2015.2
-------	--------

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the attribute to match on.
options.localName	string	Required	Local name of the attribute to match on.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: namespaceURI	The method is called with an illegal namespace value.

Syntax

```
//Add additional code
...
var attrExists = elem[0].hasAttributeNS({
  namespaceURI : '*',
  localName : 'lang'
});
...
//Add additional code
```

Element.removeAttribute(options)

Method Description	Removes the attribute with the specified name.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

Note: The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	Name of the attribute to remove.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: name	Attribute with the specified name cannot be removed.

Syntax

```
//Add additional code
...
elem[0].removeAttribute({
    name : 'lang'
});
...
//Add additional code
```

Element.removeAttributeNode(options)

Method Description	Removes the attribute specified as a xml.Attr object.
Returns	xml.Attr
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.oldAttr	xml.Attr	Required	xml.Attr object to remove.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	NOT_FOUND_ERR: An attempt is made to reference a node in a context where it does not exist.	Attribute node cannot be removed.

Syntax

```
//Add additional code
...
```

```

var removedAttr = elem[0].removeAttributeNode({
    oldAttr : attr
});
...
//Add additional code

```

Element.removeAttributeNS(options)

Method Description	Removes the attribute with the specified namespace URI and local name. <div>  Important: This method is not supported on Internet Explorer. </div>
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the attribute node to remove.
options.localName	string	Required	Local name of the attribute node to remove.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: namespaceURI	Attribute with the specified namespace cannot be removed.

Syntax

```

//Add additional code
...
elem[0].removeAttributeNS({
    namespaceURI : '*',
    localName : 'lang'
});
...
//Add additional code

```

Element.setAttribute(options)

Method Description	Adds a new attribute with the specified name. If an attribute with that name is already present in the element, its value is changed to the value specified in method argument. If an attribute with the specified name already exists, the value of the attribute is changed to the value of the value parameter.
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.name	string	Required	Name of the attribute to add.
options.value	string	Required	Value of the attribute to add.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INVALID_CHARACTER_ERR: An invalid or illegal XML character is specified.	Value for the attribute cannot be set.

Syntax

```
//Add additional code
...
elem[0].setAttribute({
  name : 'lang',
  value : 'fr'
});
...
//Add additional code
```

Element.setAttributeNode(options)

Method Description	Adds the specified attribute node. If an attribute with the same name is already present in the element, it is replaced by the new one. If an attribute with the same nodeName property already exists, it is replaced with the object in the newAttr parameter. If the attribute node replaces an existing attribute node, the method returns the new xml.Attr object.
Returns	xml.Attr

Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.newAttr	xml.Attr	Required	New xml.Attr object to add to the xml.Element object.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INUSE_ATTRIBUTE_ERR: An attempt is made to add an attribute that is already in use elsewhere.	Attribute node cannot be added.

Syntax

```
//Add additional code
...
elem[0].setAttributeNode({
    newAttr : attr
});
...
//Add additional code
```

Element.setAttributeNodeNS(options)

Method Description	<p>Adds the specified attribute node. If an attribute with the same local name and namespace URI is already present in the element, it is replaced by the new one.</p> <p>If an attribute with the same namespaceURI and localName property already exist, it is replaced with the object in the newAttr parameter. If the attribute node replaces an existing attribute node, the method returns the new xml.Attr object.</p> <div style="border: 1px solid #ffc107; padding: 5px; margin-top: 10px;">  Important: This method is not supported on Internet Explorer. </div>
Returns	xml.Attr
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.newAttr	xml.Attr	Required	New xml.Attr object to add to the xml.Element object.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INUSE_ATTRIBUTE_ERR: An attempt is made to add an attribute that is already in use elsewhere.	Attribute node cannot be added.

Syntax

```
//Add additional code
...
elem[0].setAttributeNode({
  newAttr : attr
});
...
//Add additional code
```

Element.setAttributeNS(options)

Method Description	<p>Adds a new attribute with the specified name and namespace URI. If an attribute with the same name and namespace URI is already present in the element, its value is changed to the value specified in method argument.</p> <p>If an attribute with the specified name already exists, the value of the attribute is changed to the value of the value parameter. If the attribute node replaces an existing attribute node, the method returns the new xml.Attr object.</p> <div style="border: 1px solid #f1c40f; padding: 5px; margin-top: 10px;">  Important: This method is not supported on Internet Explorer. </div>
Returns	void
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description
options.namespaceURI	string	Required	Namespace URI of the attribute node to add.

Parameter	Type	Required / Optional	Description
options.qualifiedName	string	Required	Fully qualified attribute name to add.
options.value	string	Required	String value of the attribute to add.

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	INVALID_CHARACTER_ERR: An invalid or illegal XML character is specified.	Attribute node with the specified value cannot be added.

Syntax

```
//Add additional code
...
elem[0].setAttributeNS({
  namespaceURI : '*',
  qualifiedName : 'lang',
  value : 'fr'
});
...
//Add additional code
```

Element.tagName

Property Description	The tag name of this xml.Element object.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var tagName = elem[0].tagName; \\ returns 'title'.
...
//Add additional code
```

xml.Attr

Object Description	Represents an attribute node of an xml.Element object. For a complete list of this object's properties, see Attr Object Members .
Supported Script Types	All script types

	For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var attr = elem[0].getAttributeNode({
    name : 'lang'
});
...

//Add additional code
```

Attr.name

Property Description	The name of an attribute. This property is a qualified name if the Node.localName property for the parent xml.Element object is null.
Type	string (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var attrName = attr.name; \\ returns 'lang'.
...

//Add additional code
```

Attr.ownerElement

Property Description	xml.Element object that is the parent of the xml.Attr object. Value is <code>null</code> if the attribute is not used by an element. <div>  Important: This property is not supported on Internet Explorer. </div>
Type	xml.Element (read-only)
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var attrElement = attr.ownerElement; \\ returns the title element.
...
//Add additional code
```

Attr.specified

Property Description	Returns <code>true</code> if the attribute value is set in the parsed XML document, and <code>false</code> if it is a default value in a DTD or Schema.
Type	boolean <code>true</code> <code>false</code>
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Syntax

```
//Add additional code
...
var attrSpecified = attr.specified;
...
//Add additional code
```

Attr.value

Property Description	Value of an attribute. The value of the attribute is returned as a string. Character and general entity references are replaced with their values. For example, a character reference such as <code>&#160;</code> or an entity reference such as <code>&#nbsp;</code> is replaced with a non-breaking space. <div>  Note: If you set this value, it creates a text node with the unparsed contents of the string, for example, any characters that an XML processor would recognize as markup are instead treated as literal text. </div>
Type	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Errors

Error Code	Message	Thrown If
SSS_XML_DOM_EXCEPTION	Invalid argument type, expected string: value	Cannot set the attribute value with the specified value.

Syntax

```
//Add additional code
...
var attrValue = attr.value;
...
//Add additional code
```

xml.escape(options)

Method Description	Prepares a string for use in XML by escaping XML markup, such as angle brackets, quotation marks, and ampersands.
Returns	string
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.xmlText	string	Required	String being escaped.	2015.2

Syntax

```
//Add additional code
...
var xmlEscapedDocument = xml.escape({
  xmlText : xmlFileContent
});
...
//Add additional code
```

xml.validate(options)

Method Description	<p>Validates an XML document against an XML Schema (XSD).</p> <div style="border: 1px solid #f1c40f; padding: 10px; margin: 10px 0;">  Important: This method only validates XML Schema (XSD); validation of other XML schema languages is not supported. </div> <p>The XML document must be passed as an xml.Document object. The location of the source XML Document does not matter; the validation is performed with the Document object stored in memory. The XSD must be stored in the File Cabinet.</p>
---------------------------	--

Returns	void
Supported Script Types	All server-side script types For more information, see SuiteScript 2.0 Script Types .
Governance	None
Module	N/xml Module
Since	2015.2

Parameters

 **Note:** The options parameter is a JavaScript object.

Parameter	Type	Required / Optional	Description	Since
options.xml	xml.Document	Required	The xml.Document object to validate.	2015.2
options.xsdFilePathOrId	number string	Required	The file ID or path to the XSD in the File Cabinet to validate the XML document against.	2015.2
options.importFolderPathOrId	number string	Optional	The folder ID or path to a folder in the File Cabinet containing additional XSD schemas which are imported by the parent XSD.	2015.2

Errors

Error Code	Thrown If
SSS_XML_DOES_NOT_CONFORM_TO_SCHEMA	The provided XML is invalid for the provided schema.
SSS_INVALID_XML_SCHEMA_OR_DEPENDENCY	Schema is an incorrectly structured XSD or the dependent schema cannot be found.

Syntax

```
//Add additional code
...
xml.validate({
  xml : xmlDocument,
  xsdFilePathOrId : 'SuiteScripts/schema_parent.xsd',
  importFolderPathOrId : 'SuiteScripts/'
});
...
//Add additional code
```

xml.NodeType

Enum Description	Enumeration that holds the string values for the supported node types. The Node.nodeType property is defined by one of the values in this enum.
-------------------------	---

	Use this enum to determine the type of a node in an XML document.
	 Note: Enum values are constants and therefore read-only.
Supported Script Types	All script types For more information, see SuiteScript 2.0 Script Types .
Module	N/xml Module
Since	2015.2

Values

<ul style="list-style-type: none"> ■ <code>ATTRIBUTE_NODE</code> ■ <code>CDATA_SECTION_NODE</code> ■ <code>COMMENT_NODE</code> ■ <code>DOCUMENT_FRAGMENT_NODE</code> 	<ul style="list-style-type: none"> ■ <code>DOCUMENT_NODE</code> ■ <code>DOCUMENT_TYPE_NODE</code> ■ <code>ELEMENT_NODE</code> ■ <code>ENTITY_NODE</code> 	<ul style="list-style-type: none"> ■ <code>ENTITY_REFERENCE_NODE</code> ■ <code>NOTATION_NODE</code> ■ <code>PROCESSING_INSTRUCTION_NODE</code> ■ <code>TEXT_NODE</code>
--	--	--

Syntax

```
//Add additional code
...
var DocType = xmlDocument.nodeType; \\ returns DOCUMENT_NODE
...
//Add additional code
```


SuiteScript 2.0 Record Actions and Macros

Overview of Record Action and Macro APIs

SuiteScript 2.0 supports APIs that provide the programmatic equivalent of clicking a button in the NetSuite user interface. With the record action and macro APIs, you can use SuiteScript to trigger the same business logic that is triggered by the click of a UI button. The record action and macro APIs can increase productivity by automating regular tasks that previously had to be done manually in the UI.

NetSuite records offer two alternatives for executing native NetSuite logic: a user can click a button in the UI, or a script can run that calls the API that corresponds to the button. These script and UI alternatives both produce the same results. Macro and action APIs provide ease and flexibility for your scripting. These APIs are supported for all SuiteScript 2.0 script types, both client and server-side. Macro and action APIs also can lower governance usage, because they can execute complex business logic in one API call instead of multiple API calls. For actions, governance is charged per individual action executed, and varies depending upon the type of action. For macros, no governance is charged, because changes executed by macros are saved as part of record submits.

Record actions provide a convenient way to update the state of one or more records that are in view mode. Changes that the execution of an action API makes to records are persisted in the database immediately. It is not necessary to take into account required roles, permissions, or other conditions for a record action to execute. The conditions required to execute an action are embedded in the record action API. If the conditions are not met, the action does not execute. Approve and reject are two example use cases for record actions. These actions can be applied to a single record or to multiple records of the same type, as a bulk process. When an approve or reject action is executed on a record, the approval status of the record is saved immediately.

Record macros provide an automated way to execute business logic on a record as it is edited. Changes that the execution of a macro API makes to a record are not persisted until the record is saved. An example use case for a record macro is a preview of the calculated tax amount for a sales order's items. This macro API executes after items are entered on a sales order. It results in the display of the calculated tax amount on the sales order. However, the tax amount is not saved until the record is saved. A macro API is applied only to a single record at a time. After changes to the record are saved, changes to other dependent records may occur as a result.

You need to use two different types of APIs to call record macros and actions in your scripts:

- Generic APIs to get and execute actions or macros:
 - Record action APIs are part of the [N/action Module](#). They include generic members for getting and executing actions on a record type. For details, see [N/action Module](#).
 - Record macro APIs are part of the [N/record Module](#). They include generic members for getting and executing macros on a record type. For details, see [Record Object Members](#) and [Macro Object Members](#).
- Individual APIs that implement specific logic on a specific record type:
 - A limited number of Individual actions for specific record types are supported. For details, see [Supported Record Actions](#).
 - A limited number of individual macros for specific record types. For details, see [Supported Record Macros](#).

Supported Record Actions

The following table lists currently supported record actions. They are ordered by record type and include the label on the corresponding UI button. For details about a specific action, click its ID.

Record Type	Action ID	UI Button Label
Revenue Arrangement	allocate	Allocate
Time (TimeBill)	approve	Approve
Time (TimeBill)	reject	Reject
Time (TimeBill)	submit	Submit
Weekly Timesheet	approve	Approve All Pending
Weekly Timesheet	reject	Reject All Pending
Weekly Timesheet	submit	Submit

Supported Record Macros

The following table lists currently supported record macros. They are ordered by record type and include the label on the corresponding UI button. For details about a specific macro, click its ID.

Record Type	Macro ID	UI Button Label
Billing Rate Card	modifyPriceByPercent	Recalculate
Project Charge Rule	copyResources	Copy Resources from Tasks
Weekly Timesheet	checkTimeLimits	Check Time Limits
Weekly Timesheet	copyFromWeek	Copy From Week

SuiteScript 2.0 JSDoc Validation

JSDoc 3 is a documentation generator for JavaScript source code. Users typically employ this tool to generate an HTML API reference. Developers insert specific comment blocks into their source code. These comment blocks start with `/**` and end with `*/`.

JSDoc also includes its own markup language, which is made up of JSDoc tags. These tags are prefaced with the `@` symbol. The JSDoc tags are added to the comment blocks, and are used to specify the various entities of a JavaScript file (for example, `@param`).

```
/**
 * Creates a file.
 * @param {string} name [required] - file name
 * @param {string} fileType [required] - file type
 * @param {string} contents [required] - file content
 * @returns {object} file.File
 */
```

JSDoc parses the source code for each comment block. The HTML output is then generated based on the content of the comment blocks and an evaluation of the code.

JSDoc 3 also provides users with the ability to create custom JSDoc tags. These tags can be defined to trigger events (for example, displaying a certain page).

SuiteScript 2.0 includes two custom tags, one of which is required in each entry point script uploaded to NetSuite (see [SuiteScript 2.0 JSDoc Tags](#)). When a SuiteScript 2.0 script record is requested, NetSuite uses JSDoc 3 to evaluate the entry point script and parse the code for the required JSDoc tag. This tag is used to validate the SuiteScript version.

Note: SuiteScript 2.0 users can use JSDoc 3 to create their own documentation for scripts, custom modules, and SuiteApps. To take advantage of this tool, developers must download JSDoc 3 from the official website. For additional information on JSDoc 3, see <http://usejsdoc.org/>.

JSDoc Comment Blocks

JSDoc tag comment blocks must start with a `/**` and end with a `*/` to be recognized.

Valid Example:

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

Invalid Examples:

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

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

JSDoc tags consist of a key/value pair. The key ends with the first white space after a string starting with an `@`. The value starts with the next non-whitespace character, and ends with the next carriage return. Each comment line within the block is prefixed with an `*`.

Valid Example:

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

Invalid Example:

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

SuiteScript 2.0 JSDoc Tags

The following table describes the available SuiteScript 2.0 JSDoc tags. Note that SuiteScript 2.0 entry point scripts must include two tags: @NApiVersion and @NScriptType.

JSDoc Tag	Possible Values	Required/Optional	Description
@NApiVersion	2.0 2.x 2.X	Required for entry point scripts Optional for custom modules	<p>This JSDoc tag is used in two ways:</p> <ul style="list-style-type: none"> For SuiteScript 2.0 entry point scripts, this tag is a required declaration. It signifies to NetSuite the SuiteScript version to use. For SuiteScript 2.0 custom modules that are not entry point scripts, this tag is an optional declaration. It can serve as a defense against future incompatible versions of SuiteScript (versions 3.x and higher) attempting to load it.
@NModuleScope	SameAccount TargetAccount Public	Optional	<p>This JSDoc tag is used to control access to scripts and custom modules.</p> <ul style="list-style-type: none"> If the value is set to <code>SameAccount</code>, access to the module is limited to other modules from the same bundle, and modules native to the same source account and any associated sandboxes and Release Preview accounts. Source code is not hidden at runtime. If the value is set to <code>TargetAccount</code>, access to the module is limited to other modules from the same bundle, and modules native to the same source account, target account, and any associated sandboxes and Release Preview accounts. Source code is hidden at runtime. If the value is set to <code>Public</code>, any script in the account can load and use the module.

JSDoc Tag	Possible Values	Required/Optional	Description
			Source code is hidden at runtime. The default value is <code>SameAccount</code> . For more information, see Controlling Access to Scripts and Custom Modules .
@NScriptType	BundleInstallationScript ClientScript MapReduceScript MassUpdateScript Portlet Restlet ScheduledScript Suitelet UserEventScript WorkflowActionScript	Required for entry point scripts	This tag identifies the type of script defined in the file.

Controlling Access to Scripts and Custom Modules

You can define the scope of environments (associated NetSuite accounts, sandboxes, and bundles) that may access a script. Access refers to whether the module can be loaded and invoked by another module. Modules are either loaded by the NetSuite system (via entry point modules) or by other modules (as libraries).



Important: Before deploying a SuiteApp, make sure that you review the module scope. This practice will help you to avoid:

Deploying a SuiteApp that doesn't work as expected because it can't access a necessary module in another bundle.

Providing wider access than desired to third parties and other accounts.

To set the scope for a script, you must include the appropriate value using the JSDoc tag. This tag is optional but recommended.

If you do not set the scope, `SameAccount` applies as the default. For information about adding a JSDoc tag, see [SuiteScript 2.0 JSDoc Validation](#).

The following table lists and describes the access control modifiers (supported module scopes) that are available in SuiteScript 2.0:

ModuleScope Value	Description	Access Disallowed When:	Visibility of Source Code	Examples
SameAccount	Limits script access to modules native to the same environment in which the script was created and uploaded. This environment includes the account from which a bundle is installed and can also include the related family of sandbox accounts. For more information about sandbox accounts, see the	A bundled module from a different source account attempts to import the module.	Visible during runtime	<ul style="list-style-type: none"> Installing a customization from a single sandbox to a production environment that is linked to the same account. Distributing a bundle with private business logic as an ISV (Independent Software Vendor).

ModuleScope Value	Description	Access Disallowed When:	Visibility of Source Code	Examples
	help topic Understanding NetSuite Account Types . A script file that is 'native' to the environment is added to an account using an authenticated user session.			
TargetAccount	Limits script access to modules native to the same source or target environment as the script. A source environment includes the NetSuite account (and any associated sandboxes and Release Preview accounts) from which a bundle is installed into another account. A target environment includes the NetSuite account (and any associated sandboxes and Release Preview accounts) into which a bundle is installed (whether via Bundle Copy or Bundle Install).	A bundled module that is not native to the target or source account attempts to import the module.	Hidden during runtime	<ul style="list-style-type: none"> Account administrators distributing private business logic between accounts they control, and the modules in the bundle are needed by other modules created in the target account Distributing a bundle with public APIs intended for import only by modules native to the target account.
Public	Any bundle (native and third party) that is installed in the account can run the script.	The script is not installed in the account.	Hidden during runtime	<ul style="list-style-type: none"> Customers installing customizations from multiple sandbox accounts to production, where modules from different sandboxes need to load each other. Installing a customization from a development account to a sandbox or production account. Developing open source code.

For more information about packaging and distributing bundles (SuiteApps), see the help topic [SuiteBundler](#).

NetSuite Development Accounts and Module Scope

 **Warning:** If you use a NetSuite Development account to work on a module that you intend to distribute or test, NetSuite recommends that you choose a module scope value rather than accept the default.

Keep in mind that development accounts are limited to 5 users and isolated from production and sandbox accounts. Set the value to `Public` if other bundles or accounts depend on using a module that is sourced from a development account.

If the default module scope is used on a script in a development account that is packaged into a bundle and installed to production or sandbox accounts, that script will not be accessible to modules installed

from other NetSuite accounts. For example, when developers use multiple developer accounts to collaborate on a project, the `SameAccount` module scope might not be an appropriate access control level. This scope prevents modules installed from different accounts from loading one another.

For more information about development accounts, see the help topics [The Development Account](#), [NetSuite Development Accounts](#), and the [SuiteApp Development Process with SuiteBundler](#).

SuiteScript 2.0 Entry Point Script Creation and Deployment

After you write an entry point script, you must take the following steps to run the script in your NetSuite account:

1. Make sure that the file has the required script elements, as described in [SuiteScript 2.0 Entry Point Script Validation](#), and upload the file to your File Cabinet.
2. Do one of the following:
 - Deploy your script on one or more record types, as described in [SuiteScript 2.0 Record-Level Scripts](#).
 - If your script is a client script that should be deployed only at the form level, follow the steps described in [SuiteScript 2.0 Form-Level Scripts](#).

For help understanding the difference between deploying a client script at the record level versus the form level, see [Record-Level and Form-Level Scripts](#). Note that only client scripts can be deployed at the form level. All other types of entry point scripts can be deployed at the record level only.

Record-Level and Form-Level Scripts

A client script can be deployed in one of two ways: at the record level, or at the form level.

When you deploy a client script at the record level, you deploy it globally on one or more record types. The script runs on all forms associated with the record type. By contrast, with a form-level deployment, you attach the script to a custom form associated with a record type, and the script runs only when that form is used.

For example, you could use record-level deployment to configure a client script to run on the employee record type. With this approach, the script runs whenever the employee record is used, regardless of which form is being used. With record-level deployment, it is also possible to limit the script by configuring it to be available only to certain audiences. With this approach, the script runs only when the form is used by people in certain roles, groups, or other classifications, as configured on the Audience subtab of the script deployment record. However, with record deployment, you cannot limit the script to only one form. The script behaves the same way on all forms associated with the record type.

By contrast, you could attach the client script to only one custom entry form for the employee record type. If a user then opened an employee record using the standard entry form, the script would not run. You can attach a client script to any custom entry form, custom transaction form, or custom address form. However, you cannot limit the audience for a form-level script.

Note that only client scripts can be deployed at the form level. All other entry point scripts can be deployed at the record level only.

For full details about deploying a script at the record level, see [SuiteScript 2.0 Record-Level Scripts](#). For details about deploying a client script at the form level, see [SuiteScript 2.0 Form-Level Scripts](#).

Note: Record-level client scripts can also be used on forms and lists that have been generated through [UI Objects](#) during [Suitelets](#) development. Form-based client scripts cannot be used by Suitelets.

SuiteScript 2.0 Entry Point Script Validation

For a SuiteScript 2.0 entry point script to be usable, the script must contain certain required elements. The system parses your file for these elements when you upload the file to the File Cabinet. At that time, the system also saves data about your file. This data is used later when you create a script record based on the script.

If NetSuite detects that an element within the file is missing or formatted incorrectly, it returns an error. Errors can be returned at the time you upload the file to the File Cabinet, or when you try to create a script record. If you are attaching a client script to a custom form, errors can also be returned at that time.

For more details, see the following sections:

- [Entry Point Script Validation Guidelines](#)
- [Entry Point Script Validation Examples](#)
- [Entry Point Script Validation Error Reference](#)

Entry Point Script Validation Guidelines

For an entry point script to be properly formatted, its structure must meet certain guidelines. Additionally, the script must include two required JSDoc tags.

If the script does not meet all requirements, you cannot create a script record based on the script, nor can you attach it to a form. In some cases, you are not permitted to upload the file. Similarly, if a correctly formatted script has been attached to a script record or a custom form, you are not permitted to edit the file and save changes that would introduce errors.

For details, see the following sections:

- [Entry Point Script Validation Terms and Overview](#)
- [Correct Structure for Entry Point Scripts](#)
- [Required JSDoc Tags for Entry Point Scripts](#)

Entry Point Script Validation Terms and Overview

When learning about script validation requirements and errors, it is important to understand certain terms. These terms are described in the following illustration and table.

```

1  /**
   * @NApiVersion 2.x
   * @NScriptType UserEventScript
   */
2
   define(['N/record'],
     function(record) {

3       function beforeLoad(context) {
         if (context.type !== context.UserEventType.CREATE)
           return;
         var customerRecord = context.newRecord;
         customerRecord.setValue('phone', '555-555-5555');
         if (!customerRecord.getValue('salesrep'))
           customerRecord.setValue('salesrep', 46);
       }
       function beforeSubmit(context) {
         if (context.type !== context.UserEventType.CREATE)
           return;
         var customerRecord = context.newRecord;
         customerRecord.setValue('comments', 'Please follow up with this customer!');
       }
       function afterSubmit(context) {
         if (context.type !== context.UserEventType.CREATE)
           return;
         var customerRecord = context.newRecord;
         if (customerRecord.getValue('salesrep')) {
           var call = record.create({
             type: record.Type.PHONE_CALL,
             isDynamic: true
           });
           call.setValue('title', 'Make follow-up call to new customer');
           call.setValue('assigned', customerRecord.getValue('salesrep'));
           call.setValue('phone', customerRecord.getValue('phone'));
           try {
             var callId = call.save();
             log.debug('Call record created successfully', 'Id: ' + callId);
           } catch (e) {
             log.error(e.name);
           }
         }
       }
     }

4   return {
5     beforeLoad: beforeLoad,
6     beforeSubmit: beforeSubmit,
     afterSubmit: afterSubmit
   };
});

```

Callout	Description
1	JSdoc tags
2	JSdoc tag values
3	entry point function definitions
4	interface (sometimes called script type interface)
5	entry points
6	entry point functions (sometimes called script type functions)

At a high level, the following rules apply:

- Each script must have a structure that meets certain criteria, as described in the next section, [Correct Structure for Entry Point Scripts](#).
- Each script file must include two JSdoc tags: @NApiVersion and @NScriptType. For more details, see [Required JSdoc Tags for Entry Point Scripts](#).

Correct Structure for Entry Point Scripts

An entry point script cannot be associated with a script record or custom form unless the script meets certain criteria. That is, all of the following must be true:

- The script must include one and only one properly structured define statement. The script cannot include multiple define statements.
- The define statement cannot include direct references to SuiteScript 2.0 API, objects or enums. All such references must be wrapped in a function.
- The script's return statement must include an interface whose entry points are associated with one and only one script type. Put another way, the return statement must include only one script type interface.
- The interface must include at least one entry point.
- Each entry point must correspond with an entry point function. All entry point functions must be defined in the same file.
- The script type interface implemented must match the @NScriptType value.
- JavaScript syntax errors are not permitted.

Required JSDoc Tags for Entry Point Scripts

Every entry point script must contain the JSDoc tags described in the following table. Note that this table includes details about some errors, but for full details about validation errors, see [Entry Point Script Validation Error Reference](#).

JSDoc Tag	Possible Values	Purpose	Possible Errors
@NApiVersion	2.0 2.x 2.X	Identifies the SuiteScript version to use.	If you omit this tag within a file that is properly formatted in all other ways for SuiteScript 2.0, the upload of the script fails. For details, see FAILED_TO_VALIDATE_SCRIPT_FILE . If you use an invalid value for this tag, the upload of the script fails. For details, see INVALID_API_VERSION .
@NScriptType	BundleInstallationscript ClientScript MapReduceScript MassUpdateScript Portlet Restlet ScheduledScript Suitelet UserEventScript WorkflowActionScript	Identifies the type of script defined in the file.	If you use an @NScriptType value that is incompatible with the entry points included in the script, the upload of the script fails. For details, see WRONG_SCRIPT_TYPE . If you omit this tag within a file that is properly formatted in all other ways for SuiteScript 2.0, you can upload the file, but you cannot create a script record for it, nor can you attach it to a form. For details, see FAILED_TO_VALIDATE_SCRIPT_FILE .

For full details about working with JSDoc tags in SuiteScript 2.0, including details on formatting them properly, see [SuiteScript 2.0 JSDoc Validation](#).

Entry Point Script Validation Examples

The following examples show correct and incorrect code snippets.

Correct: Includes All Required Elements

The following script is correctly formatted. It includes a define statement, a return statement with an entry point, and a function that corresponds with the entry point. Additionally, the value of the @NScriptType tag matches the script type interface used.

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

define(['N/record'],
  function(record)
  {
    function myBeforeSubmitFunction(context)
    {
      if (context.type !== context.UserEventTypes.CREATE)
        return;
      var customerRecord = context.newRecord;
      customerRecord.setValue({
        fieldId: 'comments',
        value: 'Please follow up with this customer!'
      });
    }
    return {
      beforeSubmit: myBeforeSubmitFunction
    };
  });
```

Incorrect: Missing Return Statement

The following script does not have a return statement. If you try to upload a script structured this way, the system returns an error reading "SuiteScript 2.0 entry point scripts must implement one script type function." For details, see [SCRIPT_OF_API_VERSION_20 MUST ...](#)

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

define(['N/record'],
  function(record)
  {
    function myBeforeSubmitFunction(context)
    {
      if (context.type !== context.UserEventTypes.CREATE)
        return;
      var customerRecord = context.newRecord;
      customerRecord.setValue({
        fieldId: 'comments',
        value: 'Please follow up with this customer!'
      });
    }
  }
});
```

Incorrect: Missing Define Statement

The following script uses a `require` statement instead of a `define` statement. If you try to upload a script structured like this, the system returns an error reading “SuiteScript 2.0 entry point scripts must implement one script type function.” For details, see [SCRIPT_OF_API_VERSION_20 MUST ...](#)

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

require(['N/record'],
  function(record)
  {
    function myBeforeSubmitFunction(context)
    {
      if (context.type !== context.UserEventTypes.CREATE)
        return;
      var customerRecord = context.newRecord;
      customerRecord.setValue('comments',
        'Please follow up with this customer!');
    }
  });
```

Incorrect: Missing JSDoc tag

The following script is incorrectly formatted because it does not have an `@NScriptType` tag. The system would permit you to upload a file with this script. However, if you try to create a script record with this script, the system returns an error reading “@NScriptType is mandatory for 2.0 entry point script.” For details, see [MISSING_SCRIPT_TYPE](#).

```
/**
 * @NApiVersion 2.0
 */

define(['N/record'], function(record) {
  function myBeforeSubmitFunction(context) {
    if (context.type !== context.UserEventTypes.CREATE)
      return;
    var customerRecord = context.newRecord;
    customerRecord.setValue('comments',
      'Please follow up with this customer!');
  }
  return {
    beforeSubmit : myBeforeSubmitFunction
  };
});
```

Incorrect: Missing JSDoc tag

The following script is incorrectly formatted, because it does not have an `@NApiVersion` tag. If you try to upload a script structured like this, the system returns an error reading “Failed to validate script file.” For details, see [FAILED_TO_VALIDATE_SCRIPT_FILE](#).

```

/**
 * @NScriptType usereventscript
 */

define([ 'N/record' ], function(record) {
    function myBeforeSubmitFunction(context) {
        if (context.type !== context.UserEventTypes.CREATE)
            return;
        var customerRecord = context.newRecord;
        customerRecord.setValue('comments',
            'Please follow up with this customer!');
    }
    return {
        beforeSubmit : myBeforeSubmitFunction
    };
});

```

Entry Point Script Validation Error Reference

The following table describes errors that can be returned when working with entry point scripts. These errors can be thrown during the process of uploading entry point scripts, when creating script records, or when attaching scripts to forms. Some errors can also be returned when editing script files that have already been uploaded to NetSuite, attached to script records, or attached to forms.

Error Code	Long Description
CANNOT_CHANGE_API_VERSION	This file is used by a SuiteScript {1} script; you cannot change the API version of the file.
FAILED_TO_VALIDATE_SCRIPT_FILE	Failed to validate script file: {1}
INVALID_API_VERSION	Invalid JSDoc tag value; valid values are: @NApiVersion [2.X, 2.x, 2.0]
INVALID_JSDOC_TAG_VALUE	Invalid JSDoc tag value; valid values are: {1}
MISSING_SCRIPT_TYPE	@NScriptType is mandatory for 2.0 entry point script.
MULTIPLE_DEFINE_CALLS	Invalid define call, define should only be called one time per module. Define calls found at the following line numbers: {1}
SCRIPT_OF_API_VERSION_20_CANNOT_IMPLEMENT_MORE_THAN_ONE_SCRIPT_TYPE_INSTANCE (See SCRIPT_OF_API_VERSION_20 CANNOT ...)	Script of 2.0 entry point scripts cannot implement functions for more than one script type.

Error Code	Long Description
SCRIPT_OF_API_VERSION_20_MUST_IMPLEMENT_A_SCRIPT_TYPE_INTERFACE (See SCRIPT_OF_API_VERSION_20 MUST)	SuiteScript 2.0 entry point scripts must implement one script type function.
SS_V2_FILE_USED_FOR_FORM_SCRIPTS_MUST_IMPLEMENT_CLIENT_SCRIPT_TYPE (See SS_V2_FILE_USED_FOR_FORM_SCRIPT)	SuiteScript version 2 file used for form scripts must implement client script type
SYNTAX_ERROR	Syntax error: {1}
THE_FILE_IS_USED_AS_SCRIPT_TYPE_1_CANNOT_BE_CHANGED_TO_2 (See THE_FILE_IS_USED_AS_SCRIPT_TYPE_1 ...)	The file is used as script type {1}, cannot be changed to {2}
WRONG_SCRIPT_TYPE	Script file includes @NScriptType {1}; this script type cannot be used to {2}.

CANNOT_CHANGE_API_VERSION

The long description for this error code is: This file is used by a SuiteScript {1} script; you cannot change the API version of the file.

This error can be displayed when you attempt to edit a script file associated with a script record or a custom form. Specifically, this error is displayed if you try to edit and save changes to the @NApiVersion value. For example, if you try to remove the expression @NApiVersion 2.0 from a version 2.0 script, the system displays this error. Similarly, if you try to add @NApiVersion 2.0 expression to a version 1.0 script, the system displays this error.

In some cases, this error is preceded by the [INVALID_API_VERSION](#) error. The INVALID_API_VERSION error is included if the new @NApiVersion value you are trying to use is invalid.

Note that SuiteScript 1.0 files do not use the @NApiVersion tag. For help creating and uploading SuiteScript 1.0 files, see the help topic [Running Scripts in NetSuite Overview](#).

FAILED_TO_VALIDATE_SCRIPT_FILE

The long description for this error code is: Failed to validate script file: {1}

The system displays this error in a few cases. For example, you see this message if your script is formatted correctly for SuiteScript 2.0 in all ways except that it lacks the @NApiVersion tag.

INVALID_API_VERSION

The long description for this error code is: Invalid JSDoc tag value; valid values are: @NApiVersion [2.X, 2.x, 2.0]

This error is displayed when you attempt to upload a file with an invalid value for the @NApiVersion tag. It can also be displayed if you edit a script file that was already uploaded and try to modify the value of @NApiVersion tag.

If you see this error, check your file to make sure the @NApiVersion tag has a valid value. Valid values include the following:

- 2.0

- 2.x
- 2.X

INVALID_JSDOC_TAG_VALUE

The long description for this error code is: Invalid JSDoc tag value; valid values are: {1}

This error can be displayed if your script file uses an invalid value for any JSDoc tag. For example, if you use an invalid value for @NScriptType, the system displays this error.

To resolve the problem, check your file to make sure the value you used for the tag does not include typos or other errors.

MULTIPLE_DEFINE_CALLS

The long description for this error code is: Invalid define call, define should only be called one time per module. Define calls found at the following line numbers: {1}

This error is displayed if your script includes more than one define call. If you see this error, review the script and edit it appropriately.

MISSING_SCRIPT_TYPE

The long description for this error code is: @NScriptType is mandatory for 2.0 entry point script.

The system displays this error if you attempt to create a script record based on a SuiteScript 2.0 script that does not include the @NScriptType tag. To resolve the problem, edit the file and add the @NScriptType tag with the appropriate value.

The system does not display this error when you are uploading your file to the File Cabinet. It displays this error only when you try to create a script record, or if you edit a script file attached to a script record.

SCRIPT_OF_API_VERSION_20 CANNOT ...

The full code for this error is:

SCRIPT_OF_API_VERSION_20_CANNOT_IMPLEMENT_MORE_THAN_ONE_SCRIPT_TYPE_INTERFACES

The corresponding long description is: SuiteScript 2.0 entry point scripts cannot implement functions for more than one script type.

The system displays this error if your interface includes entry points from more than one script type. If you see this error, review your script's interface. Check that all entry points belong to a single script type. For details on the available entry points for each script type, see [SuiteScript 2.0 Script Types](#).

SCRIPT_OF_API_VERSION_20 MUST ...

The full code for this error is:

SCRIPT_OF_API_VERSION_20_MUST_IMPLEMENT_A_SCRIPT_TYPE_INTERFACE.

The corresponding long description is: SuiteScript 2.0 entry point scripts must implement one script type function.

This error signifies that your script is missing an interface or that an error exists within the interface. This error can be returned when you try to upload a file, or when you try to edit a file that was previously uploaded.

SS_V2_FILE_USED_FOR_FORM_SCRIPT

The full code for this error is:

SS_V2_FILE_USED_FOR_FORM_SCRIPTS_MUST_IMPLEMENT_CLIENT_SCRIPT_TYPE.

The corresponding long description is: SuiteScript version 2 file used for form scripts must implement client script type.

This error is displayed if you try to attach a script other than a client script to a custom form. Only a client script can be deployed at the form level. Other types of entry point scripts can be deployed only at the record level. For full details about working with form scripts, see [SuiteScript 2.0 Form-Level Scripts](#). For details about deploying other types of entry point scripts, see [SuiteScript 2.0 Record-Level Scripts](#).

SYNTAX_ERROR

The long description for this error code is: Syntax error: {1}

THE_FILE_IS_USED_AS_SCRIPT_TYPE_1 ...

The full error code for this error is:

THE_FILE_IS_USED_AS_SCRIPT_TYPE_1_CANNOT_BE_CHANGED_TO_2.

The corresponding long description is: The file is used as script type {1}, cannot be changed to {2}

You may see this error when editing a script file that is attached to an existing script record. If you edit the file so that it uses a different script type interface from what it previously used, and a different @NScriptType value, the system generates this error.

This error occurs because the Type field on the script record cannot be changed. To avoid this problem, make your changes in a file that is not already associated with a script record. Then create a new script record based on that file.

WRONG_SCRIPT_TYPE

The long description for this error code is: Script file includes @NScriptType {1}; this script type cannot be used to {2}.

The system displays this error if the file @NScriptType value is not compatible with the entry points in the script's return statement. If you see this error, double-check the @NScriptType value used in your script. Make sure that it corresponds with the entry points used by your script. If the tag's value does not match the script type interface, you cannot upload the file to the File Cabinet. For details on the available entry points for each script types, see [SuiteScript 2.0 Script Types](#).

SuiteScript 2.0 Record-Level Scripts

After you have written a valid entry point script, as described in [SuiteScript 2.0 Entry Point Script Validation](#), you must take the following steps if you want to deploy the script on records in your NetSuite account:

1. Use the script file to create a script record, as described in [Script Record Creation](#). The script record identifies the script's internal ID, whether or not the script is active, and other details.
2. Deploy the script, as described in [Script Deployment](#). You use script deployments to configure details regarding how and when the script runs. The types of choices you make vary depending on the script's type.

Important: These steps are specific to entry point scripts deployed at the record level.

Script Record Creation

After you have written an entry point script, you can create a script record for it. You must create a script record before you can deploy your script.

When you create a script record, you set some fields manually. By contrast, the system uses the data in the file to automatically set several key fields. These fields are described in the following table.

Field	Value Taken From
API Version	The @NApiVersion tag in your script file
Script File	The name of your script file.
Type	The @NScriptType tag in your script file
Functions (on the Script subtab)	The entry points defined within your script.

Note: Your script must be properly structured and annotated before it can be used to create a script record. For details, see [SuiteScript 2.0 Entry Point Script Validation](#).

At a high level, you can create a script record by using the following approach: Go to Customization > Scripting > Scripts > New. Select the appropriate file in the Script File dropdown list, then click **Create Script Record**.

In response, the system opens a new script record. The Type, API Version, and Script File fields are already populated. To complete the process, enter a name for the script record, and set any optional fields as needed. Then click **Save**.

After you save, the system shows the new script record in view mode. On the Scripts subtab, the system lists all possible entry point functions that could exist for this script type, with checks beside the ones used in your script.

Script

Edit
Back
Deploy Script
Download XML
Actions ▾

TYPE

User Event

NAME

New User Event Script

PACKAGE

ID

customscript_usereventscript

API VERSION

2.0

DESCRIPTION

OWNER

John Smith

☐ INACTIVE

Scripts
Parameters
Unhandled Errors
Execution Log
Deployments
System Notes

SCRIPT FILE

preview
newUserEventScript.js
download
Edit

☒ BEFORE LOAD FUNCTION
 ☒ BEFORE SUBMIT FUNCTION
 ☒ AFTER SUBMIT FUNCTION

The check boxes on the Scripts subtab cannot be edited directly. To check or clear the boxes, edit the script to add or remove the appropriate functions. When you save your changes, the script record's check boxes are updated to match the contents of the script file.

For more detailed help on creating a script record, see [Creating a Script Record](#).

Creating a Script Record

The process of creating a script record varies depending on the script's type. The following procedure uses general steps that apply to all types. For certain script types, additional steps may be required.

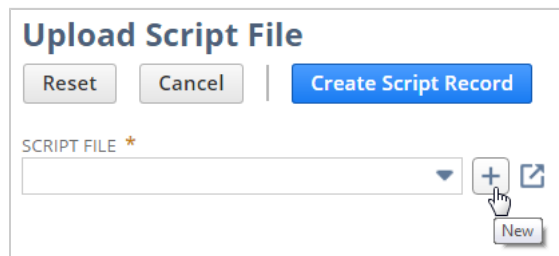
To create a script record:

1. Go to Customization > Scripting > Scripts > New.

The system displays the **Upload Script File** page.

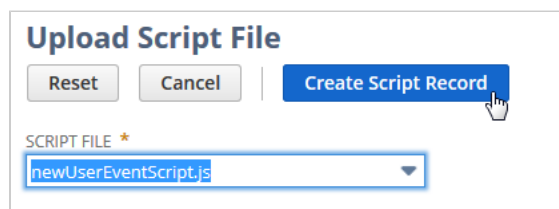
2. In the **Script File** dropdown list, select the appropriate SuiteScript 2.0 file. The dropdown list populates automatically with SuiteScript files that you have uploaded to your File Cabinet. This list includes version 1.0 and version 2.0 files.

If the file you want to use has not been uploaded yet, you can upload it from this page. Point to the area at the right of the dropdown list to display a Plus icon.



Click the icon to display a dialog box for uploading a file from your local environment.

3. After you have populated the dropdown list, click the **Create Script Record** button.



The system displays the **Script** page, with your .js file listed on the **Scripts** subtab. The read-only **Type** field is automatically populated based on the content of your file. The read-only **API Version** field is automatically populated with the version number: **2.0**.

Note: If the system redirects you to a page that prompts you to select a script type, that means that in Step 2 you identified a SuiteScript 1.0 file. You can click the Back button in your browser to return to the previous page and select a different file. Or, for details on creating a script record for SuiteScript 1.0, see the help topic [Steps for Creating a Script Record](#).

4. Enter a name in the **Name** field.
5. In the **ID** field, optionally enter a custom ID for the script record. If the **ID** field is left blank, a system-generated internal ID is created for you when you save the record.
6. If the **Portlet Type** field is displayed, select the appropriate value. This field is available only for the portlet script type. In these cases, it is a required field.
7. In the **Description** field, optionally enter a description for the script.
8. If appropriate, change the value of the **Owner** field. By default, this field is set to the currently logged-in user.

After the script record is saved, only the owner of the record or a system administrator can modify the record.

9. If appropriate, check the **Inactive** box. When a script is set to Inactive, any deployments associated with the script are also inactive.
10. On the **Parameters** subtab, define any parameters (custom fields) that are used by functions in the script.
11. On the **Unhandled Errors** subtab, optionally define the people to be notified if script errors occur.

Three types of error notifications are sent:

- An initial email is sent about the first occurrence of an error within an hour.
- An email with aggregated error information for every hour is sent. (The error counter is reset when this email is sent.)
- An email is sent about the first 100 errors that have occurred after the error counter is set to 0.

For example, an error is thrown 130 times within an hour. An initial email is sent. After the 100th occurrence, another email is sent. Since there are an additional 30 occurrences within the same hour, a final summary email is sent at the end of the hour. During the second hour, if there are only 50 occurrences of the error, only one summary email is sent at the end of that hour.

 **Note:** By default, the **Notify Script Owner** box is checked. Alternatively or in addition to the script owner, you can identify other people, as follows:

- Check the **Notify All Admins** box, if all administrators should be notified.
 - Select one or more groups from the **Groups** dropdown list. To define new groups, go to Lists > Relationships > Groups > New.
 - Enter the email addresses of any other users who should be notified. You can enter a comma-separated list of email addresses.
12. Optionally, define a deployment for the script record by clicking the **Deployments** subtab and adding a line to the sublist. For details on adding a line to this list, see [Deploying a Script by using the Deployments Sublist](#). If you want to add a deployment, another option is to skip the Deployments subtab and, in the last step in this procedure, click Save and Deploy.
 13. If this is a client script, optionally add lines to the **Buttons** sublist, which appears on the **Scripts** subtab.
 14. If the **Scripts** subtab includes the **Custom Plug-in Types** sublist, optionally add lines to this list.
 15. To save the new record, click one of the following
 - **Save** – to save and close the record.
 - **Save and Deploy** – to save the script record and open a page that lets you create a new script deployment record.

Script Deployment

Before an entry point script will run in your NetSuite account, it must be deployed. You can deploy a script at the time you create a script record, or you can deploy it later. The deployment settings available vary depending on the script's type and on how you deploy the script.

When you deploy a script, the system creates a script deployment record. These records are listed at Customization > Scripting > Script Deployments. Deployments are also represented as lines on the Deployments subtab of the script record.

Multiple deployments can be created for the same script record. When multiple deployments exist, they are executed in the order in which they are listed on the Deployments sublist. This sequence typically corresponds with the order in which the deployments were created.

For more details, see the following topics:

- [Methods of Deploying a Script](#)
- [Deploying a Script by using the Deployments Sublist](#)
- [Updating a Script Deployment](#)
- [Managing Web Store Performance Impact](#)

Methods of Deploying a Script

To run an entry point script in your NetSuite account, you must create a script deployment for it. The deployment determines how and when the script runs. You can create a script deployment in any of the following ways. In most cases, you can also use these methods to edit a script deployment:

- [Work with the Deployments Sublist](#)
- [Use the Script Deployment Record Form](#)

- [Use N/record Module Methods](#)

Work with the Deployments Sublist

You can deploy a script at the same time as you create a script record. This process is described in [Deploying a Script by using the Deployments Sublist](#). This approach lets you define values for many deployment fields, although not all. You can use this technique at the following times:

- When you are creating a script record.
- For some script types, when you are editing an existing script record. However, for some script types, the Deployments sublist is read-only when you are editing the script record.

Use the Script Deployment Record Form

As an alternative, you can deploy a script by using the entry form for the script deployment record. You may want to use this approach for the following reasons:

- The script deployment record lets you access a greater number of fields than the Deployments sublist.
- If you have already created a script record, in some cases it is not possible to edit the Deployments sublist.

You can open the script deployment record entry form by using any of the following methods:

- View a script record, and click the **New Deployment** button.
- View an existing script deployment record, and select **Actions > New**.
- View the list of a script record's existing deployments, and click the **New Deployment** button. (To display the list of deployments, go to Customization > Scripting > Scripts, and click the Deployments link for the appropriate script record.)

Many of the body fields available during this process are the same as those you see when working with the Deployments sublist. For help with these fields, see [Deploying a Script by using the Deployments Sublist](#). For help with additional fields that are available, refer to the [SuiteScript Records Browser](#).

Use N/record Module Methods

You can create a deployment programmatically by using the [record.create\(options\)](#) method. When creating a script deployment record, set the options.type parameter to record.Type.SCRIPT_DEPLOYMENT. Similarly, if you can want to modify a script deployment record programmatically, you can load it by using [record.load\(options\)](#). You can modify the record by using other [N/record Module](#) methods.

For help with the field IDs available on the script deployment record, refer to the [SuiteScript Records Browser](#).

Deploying a Script by using the Deployments Sublist

If you want an entry point script to execute in NetSuite, you must deploy it.

This topic describes how to deploy a script by using the Deployments sublist. You can use this approach when you are in the process of creating a script record that you want to deploy. Depending on the script type, you can also use this approach when you are editing an existing script record.

 **Note:** For information on other ways of deploying a script, see [Methods of Deploying a Script](#).

Be aware that the script deployment process varies somewhat depending on the script type. The following procedure describes basic steps. For certain script types, additional steps may be required.

To deploy a script by using the Deployments sublist:

1. If the script record is not already open for editing, open it. Go to Customization > Scripting > Scripts. Locate the script for which you want to create a script deployment. Click the corresponding **Edit** link.
2. Click the **Deployments** subtab.
3. Add a value to the sublist, as follows:
 - If the **Title** column is displayed, enter a title.
 - If the **Applies to** column is displayed, select the appropriate value in the dropdown list. Specifically, select the record type where you want to deploy the script. To deploy the script on all record types supported in SuiteScript, select **All Records**.
 - Enter a meaningful name in the **ID** column. The record's ID lets you work with the deployment programmatically. Note that the system automatically adds a prefix of **customdeploy** to the value you enter. If you do not specify an ID, a system-generated ID is created for you.
 - Check or clear the **Deployed** box, as appropriate. For most script types, the default value is **Yes**. However, for map/reduce and scheduled scripts, the default is **Not Scheduled**.
 - If the **Execute in Commerce Context** column is displayed, either leave the field cleared or click in the column to display a check box. This option is available for user event scripts only. It determines whether the script will be triggered by activity in the web store. For more details, see [Managing Web Store Performance Impact](#).
 - In the **Status** column, select the appropriate deployment status. Note that the available values vary depending on the script type.
 - If the **Event Type** dropdown list is displayed, optionally select a value from the dropdown list. This value identifies the event type that triggers the script execution.
 - Optionally, in the **Log Level** field, specify which type of log messages will appear on the **Execution Log** tab when the script is executed.
 - If the **Execute as Role** column appears, optionally select whether you want the script to execute using Administrator privileges, regardless of the permissions of the currently logged-in user.
 - Click **Add** to add the new line to the sublist.
4. Click **Save**.

The system saves the deployment. If you view the Deployments subtab again, the deployment you just created is represented as a link. You can click the link to view the script deployment record that has just created been created. In some cases, this page lets you configure additional fields. For details, see [Updating a Script Deployment](#).

Updating a Script Deployment

If appropriate, you can edit an existing script deployment and make changes to it.

For some script types, you can make changes by editing the script record's Deployments sublist. However, for other types, the Deployments sublist is read-only. For these script types, you must open the full script deployment record for editing. You can do this programmatically, or by using the script deployment record entry form. This topic describes the latter method.

When you open the full script deployment record for editing, you have access to a greater number of fields than are available on the Deployments sublist. For example, the script deployment record includes the following subtabs:

- **Audience** subtab – When a script is deployed, it runs only in the accounts of the specified audience. This subtab lets you specify the roles that make up the script audience. In SuiteScript 2.0, this subtab is available for the following script types: client, portlet, RESTlet, Suitelet, and user event.
- **Links** subtab – For a Suitelet, lets you specify the menu paths that permit users to access the Suitelet.
- **Schedule** – for a map/reduce or scheduled script, lets you configure the schedule that determines when the script runs.



Important: If the script associated with the deployment is running in NetSuite, you cannot edit the deployment. You must wait until the script stops running.

To update a script deployment:

1. Go to Customization > Scripting > Script Deployments.
2. Locate the deployment you want to edit, and click the corresponding **Edit** link.
3. Make the appropriate changes.
4. Click **Save**.

Managing Web Store Performance Impact

When deploying a user event script, you can permit or prevent the deployment from being triggered by web store activity. You configure this behavior by using the script deployment record's **Execute in Ecommerce Context** option. This option lets you control whether the deployment is triggered by an event in SuiteCommerce Advanced, SuiteCommerce Site Builder, or SuiteCommerce InStore.

Scripts can significantly slow web store performance. By disassociating scripts from web store activity, you can improve web store response times.

You can set this option at the time you deploy a script. You can also go back and configure the option later, as described in the following procedure.

To enable or disable web store triggers for a user event script deployment:

1. Open the deployment for editing. For example, navigate to Customization > Scripting > Script Deployments. Locate the appropriate deployment and click the corresponding edit link.
2. On the script deployment record, locate the **Execute in Ecommerce Context**. Check or clear the box as appropriate.
3. Save the record.



Note: Another option for optimizing web store performance is the **Asynchronous afterSubmit Sales Order Processing** feature. When you enable this feature, all afterSubmit user events and workflows triggered by web store checkout run asynchronously. For details, see the help topic [Ecommerce Features](#).

Viewing System Notes

If appropriate, you can review details about the history of a script or script deployment record. These details identify the user who created the record, the context in which the record was created, and the date of the last change to the record.

To view a script deployment's history

1. Do one of the following:
 - To find a script, go to Customization > Scripting > Scripts.
 - To find a script deployment, go to Customization > Scripting > Script Deployments.
2. Locate the record you want to view, and click the corresponding **View** link.
3. Click the **System Notes** subtab.

Note: In previous releases, details about these records was listed on its **History** subtab. However, that subtab is no longer updated. New activity is captured on the record's System Notes subtab.

SuiteScript 2.0 Form-Level Scripts

In some cases, you might want a client script to deploy on one form only. To configure this behavior, you attach the script to the form. You can attach a script to a custom entry form, a custom transaction form, or a custom address form.

With both custom entry forms and custom transaction forms, you can also using logic in the script to create a button or a menu item. These controls are sometimes referred to as custom actions. For custom address forms, you can deploy the script on the form, but you cannot configure custom actions.

The alternative to deploying your script on a form is to deploy it on a record type. In the latter case, the script is executed globally on that record type, regardless of which form is being used. By contrast, when you attach a client script to a form, it runs on that form only. For more details about the differences between these two approaches, see [Record-Level and Form-Level Scripts](#).

For details on deploying a client script at the form level, see the following sections:

- [Attaching a Client Script to a Form](#)
- [Configuring a Custom Action](#)

Important: Only client scripts can be attached to forms. Other types of entry point scripts can be deployed only at the record level. A script deployed at the record level is used on all forms associated with that record type. For help with record deployment, see [SuiteScript 2.0 Record-Level Scripts](#).

Attaching a Client Script to a Form

You can attach a client script to a form after you have made sure that it is a valid script, as described in [SuiteScript 2.0 Entry Point Script Validation](#).

After you attach a script to a form, it is deployed on that form and will run when the form is used. You can attach a client script to a custom entry form, a custom transaction form, or a custom address form.

Important: Users must have at least the Edit level of the SuiteScript permission to attach a script to a custom form by editing the Custom Code tab of the form record. Users with the View level of the SuiteScript permission can see the Custom Code tab, but they cannot edit it.

To attach a client script to a form:

1. Navigate to the form to which you want to attach the script. For example, go to Customization > Forms > Entry Forms. Location the appropriate form and click the corresponding **Edit** link.

2. Navigate to the **Custom Code** subtab.
3. In the **Script File** dropdown list, select the SuiteScript 2.0 script that you want to attach. If the file you want to use has not yet been uploaded to the File Cabinet, you can upload it from this page. To upload a file, point to the area at the right of the dropdown list to display a Plus icon. Click this icon to display a dialog box. You can use this dialog box to upload a file from your local environment.

After you populate the Script File field, the system updates the page to populate the **SuiteScript API Version** field. The page also updates to include a list of all possible client script entry point functions. Check marks are displayed next to the functions that are used by your script.

The screenshot shows the 'Custom Code' subtab in the SuiteScript configuration interface. It features a 'SCRIPT FILE' dropdown menu with 'VendorPageInit.js' selected. Below this is the 'SUITESCRIPT API VERSION' field, set to '2.0'. A list of functions follows, each with a checkbox indicating its status: 'PAGE INIT FUNCTION' is checked, while 'SAVE RECORD FUNCTION', 'VALIDATE FIELD FUNCTION', 'FIELD CHANGED FUNCTION', 'POST SOURCING FUNCTION', 'LINE INIT FUNCTION', 'VALIDATE LINE FUNCTION', 'VALIDATE INSERT FUNCTION', 'VALIDATE DELETE FUNCTION', and 'RECALC FUNCTION' are unchecked.

The check boxes on the **Custom Code** subtab cannot be edited directly. To check or clear the boxes, edit the script to add or remove functions as appropriate. When you save your changes, the script record's check boxes are updated to match the contents of the script file.

Note: If you edit the **Script File** field later and select a version 1.0 file instead of a version 2.0 file, the page reloads and includes fields into which you must manually enter the names of your entry-point functions. For details on this process, see the help topic [Step 3: Attach Script to Form](#).

4. Click **Save**.

Configuring a Custom Action

In some cases, you might want to configure a custom action that uses logic contained in your client script. A custom action can be either of the following:

- A custom button that appears at the top of the page.

The screenshot displays a NetSuite form for a 'Sales Order'. At the top, there are buttons for 'Save', 'Cancel', 'Auto Fill', 'Reset', and 'Send for Auditing'. The 'Send for Auditing' button is highlighted with a red rectangle. Below these buttons is a section titled 'Primary Information' containing a 'CUSTOM FORM' dropdown menu (set to 'Custom Sales Order') and a 'START DATE' input field.

- A custom menu item that appears as an option when the user points to the Actions label.

The screenshot shows a web form titled "Sales Order" with a search icon. At the top, there are buttons for "Save", "Cancel", "Auto Fill", and "Reset". To the right of these buttons is an "Actions" label with a dropdown arrow. A red box highlights the "Actions" label and its dropdown menu, which contains three items: "Send for Auditing", "Memorize", and "Go To Register". Below the buttons, the form has a "Primary Information" section. It includes a "CUSTOM FORM" dropdown menu currently set to "Custom Sales Order", an "ORDER #" field with the value "SORD10073", and two date fields labeled "START DATE" and "END DATE".

You configure these elements by using logic contained in the client script attached to your form.

Note: It is not possible to configure a custom action for a custom address form. You can configure custom actions for custom entry and custom transaction forms.

To configure a custom action on a custom form:

1. If the form is not already open for editing, open it. For example, go to Customization > Forms > Transaction Forms. Locate the appropriate form and click the corresponding **Edit** link.
2. Navigate to the **Actions** subtab.
3. Navigate to the **Custom Actions** subtab.
4. In the **Label** column, enter a label for your button or menu item.
5. In the **Function** column, enter the name of the appropriate entry point function from your client script.
6. In the **Display as** column, select **Button** or **Menu** as appropriate.
7. Click **Save**.

Note: For more information about both custom actions and standard actions, see the help topic [Configuring Buttons and Actions](#). For help understanding the validation terms used in this topic, see [Entry Point Script Validation Guidelines](#).

SuiteScript 2.0 Custom Modules

- [Custom Modules Overview](#)
- [Writing a Custom Module](#)
- [Preparing to Add a Custom Module](#)
- [Module Dependency Paths](#)
- [Naming a Custom Module](#)
- [Custom Module Examples](#)
- [Troubleshooting Errors](#)
- [Frequently Asked Questions: Custom Modules](#)

Custom Modules Overview

With SuiteScript 2.0, you have the ability to create custom modules (including third-party, AMD, and non-AMD modules). This supports modular and well-structured organization of code that is easier to read and modify. It also lets you build and access additional, customized API functionality.

Build a custom module to help you do the following:

- Group reusable functions into a common library file. Supporting code (such as helper functions and custom APIs) can be organized into reusable sections. These custom modules are loaded within your entry point script.
- Add custom modules to SuiteApps and expose those modules to third parties.
- Import a third-party API.
- Organize and separate utilities from business logic.

Writing a Custom Module

From the source suitescript file that defines your custom module, you will need to use the [define Object](#). Then, to load and use the module, specify the dependency with a [require Function](#) from an entry point script. You must also define a function that instantiates the module.

If desired, you can add standard and custom JSDoc tags to your custom module scripts. JSDoc tags are optional (unless the script is an entry point script). However, NetSuite recommends using the @NApiVersion and @NModuleScope tags in your custom modules. The SuiteScript API version tag can help protect against loading from future incompatible versions of SuiteScript (versions 3.x and higher). The module scope tag determines whether other scripts and accounts can access the custom module.

For custom JSDoc tags, SuiteScript 2.0 users can use JSDoc to create their own documentation for scripts, custom modules, and SuiteApps. To take advantage of this tool, developers must download JSDoc from the official website. For additional information on JSDoc, see <http://usejsdoc.org/>.

For script samples, see [Custom Module Examples](#).

To learn about setting up a custom module name, see [Naming a Custom Module](#).

Preparing to Add a Custom Module

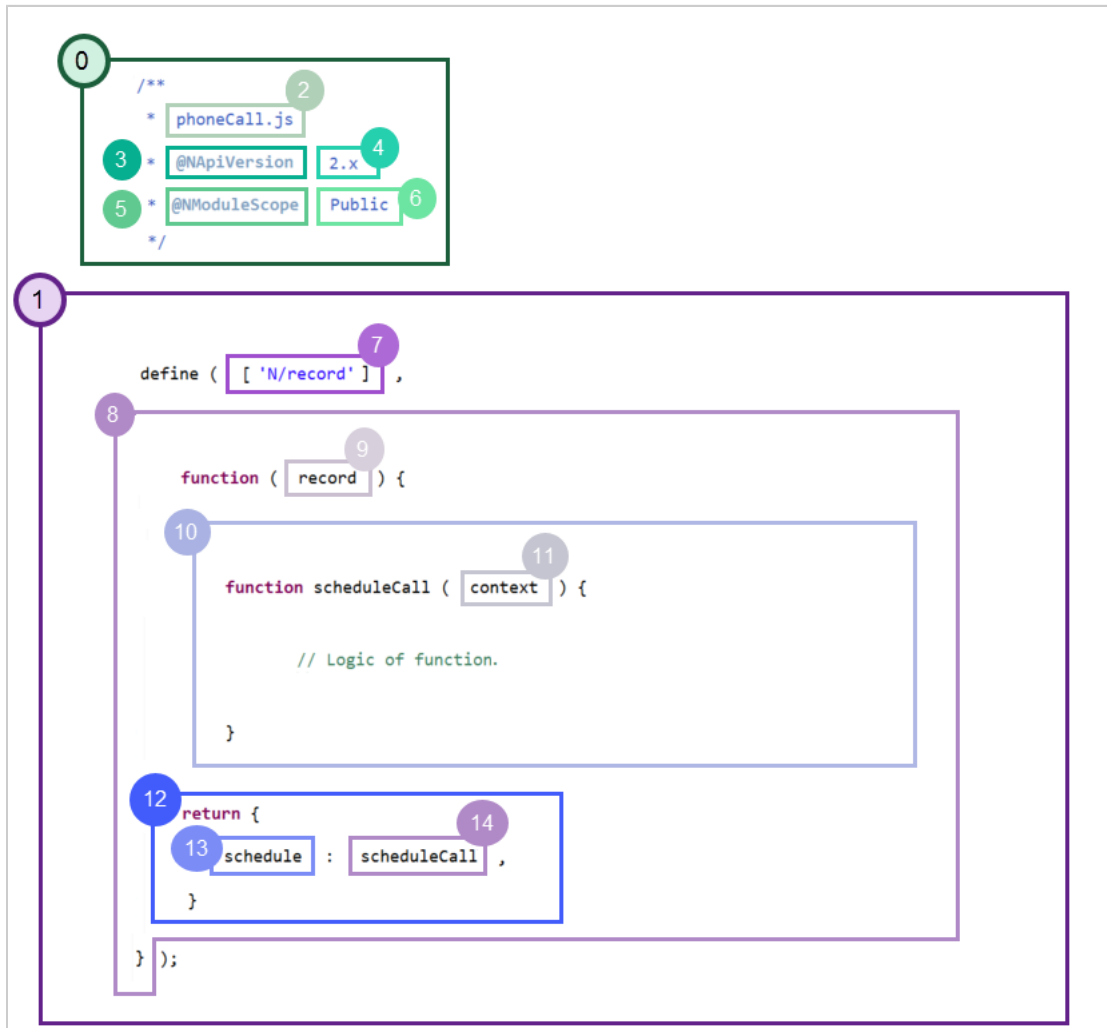
Before you work on a custom module, make sure you're familiar with the following topics:

- [SuiteScript 2.0 Script Basics](#)

- SuiteScript 2.0 Entry Point Script Validation
- SuiteScript 2.0 Global Objects and Module Dependency Paths
- Controlling Access to Scripts and Custom Modules

SuiteScript 2.0 Anatomy of a Custom Module Script

All SuiteScript 2.0 custom module scripts must conform to the same basic structure. The following diagram illustrates that structure. For an explanation of the numbered components of the script, refer to the table that follows the diagram.



General Area	Callout	Description
0 — JSDoc tags and other annotations	2	The title of the file that holds this script. This annotation is not required, but it can be useful. Any script that loads this module must refer to this name.
	3 and 4	The <code>@NApiVersion</code> tag and its value. This tag is not required in a custom module script, but you may want to include it to prevent compatibility problems with scripts that use future versions of SuiteScript.

General Area	Callout	Description
	5 and 6	The @NModuleScope tag and its value. This tag is optional. You can use it to limit the access that other scripts have to this module. For more information about this tag, see Controlling Access to Scripts and Custom Modules .
1 — define statement	7	The define function's first argument, which is a list of modules required by the script. This script uses only one: the N/record Module , which lets the script interact with records.
	8	The define function's second argument, which is a callback function.
	9	The arguments of the callback function. Because this script loads only one module, the callback function takes only one argument. This argument represents the N/record Module and can be used anywhere in the callback function to access the module's APIs. You can give this object any name you prefer but, as a best practice, use a name that is similar to the module name.
	10	The script's entry point function. As you can see in the return statement (Callout 12), this entry point function is associated with the entry point named schedule.
	11	The context object that is made available when the schedule entry point is invoked. The values of this object's properties are defined in the script that calls this custom module. For an example, see SuiteScript 2.0 Custom Module Tutorial .
	12	The callback function's return statement.
	13	The custom module's entry point. As with an entry point script, every custom module script must use at least one entry point. The difference is that an entry point script must use a standard entry point that is part of the script type identified by the @NScriptType tag. A custom module entry point is your creation, so you can use this return statement to give the entry point any name you prefer.
	14	The entry point function returned when the schedule entry point is invoked. For each entry point used, your custom module script must identify some object, usually a function, that is returned.



Note: For details on the standard structure for an entry point script, see [SuiteScript 2.0 Anatomy of a Script](#).

SuiteScript 2.0 Custom Module Tutorial

A custom module script holds logic that can be used by other scripts. With this feature, if you have a piece of logic that is required by multiple scripts, you can create a custom module script to hold that logic. This approach is more efficient than copying the logic into each script where it is needed.

This topic walks you through the process of creating a custom module script and modifying an entry point script so that it uses the custom module.

See the following sections:

- [Sample Script Overview](#)
- [Step One: Deploy the Prerequisite Script](#)
- [Step Two: Create the Custom Module Script File](#)
- [Step Three: Review the Script \(Optional\)](#)
- [Step Four: Upload the Custom Module Script File to NetSuite](#)
- [Step Five: Modify the Entry Point Script](#)
- [Step Six: Upload the Revised User Event Script](#)

■ Step Seven: Test the Script

Sample Script Overview

The custom module script in this topic can be used to automatically create a phone call record. After you create this module and upload it to your NetSuite File Cabinet, you can call this module from an entry point script.

When you have finished the updates to the entry point script as described in this topic, the custom module script is triggered each time a new employee record is created. If the new employee record includes values in the Phone and Supervisor fields, the custom module schedules a phone call between the supervisor and the new employee.

Step One: Deploy the Prerequisite Script

If you have not deployed the user event script described in [SuiteScript 2.0 User Event Script Tutorial](#), deploy it now.

Step Two: Create the Custom Module Script File

Copy and paste the following code into the text editor of your choice. Save the file and name it `phoneCall.js`.

```
/**
 * phoneCall.js
 * @NApiVersion 2.x
 * @NModuleScope Public
 */

// This script must create a record, so it loads the
// N/record module.

define ( ['N/record' ] ,

    // The next line marks the beginning of the callback
    // function. The 'record' argument is an object that
    // represents the record module.

    function (record) {

        // The next line marks the beginning of the entry point
        // function.

        function scheduleCall (context) {

            var newPhoneCall = record.create({
                type: record.Type.PHONE_CALL,
                isDynamic: true
            });
        }
    }
});
```

```

        newPhoneCall.setValue({
            fieldId: 'title',
            value: context.phoneCallTitle
        });

        newPhoneCall.setValue({
            fieldId: 'assigned',
            value: context.phoneCallOwner
        });

        newPhoneCall.setText({
            fieldId: 'phone',
            text: context.phoneNumber
        });

        try {

            var newPhoneCallId = newPhoneCall.save();

            log.debug({
                title: 'Phone call record created successfully',
                details: 'New phone call record ID: ' + newPhoneCallId
            });

        } catch (e) {
            log.error({
                title: e.name,
                details: e.message
            });
        }

        // Add the return statement that identifies the entry point funtion.

        return {
            schedule: scheduleCall,
        }

    });

```

Step Three: Review the Script (Optional)

If you want to understand more about how this script is structured, review the following subsections. Note that these images do not show the entire script. For more details, refer to the comments in [Step Two: Create the Custom Module Script File](#).

JSDoc Tags

A custom module script is not required to have JSDoc tags, but they are recommended. The following illustration shows this sample's JSDoc block.


```
/**
 * phoneCall.js
 * @NApiVersion 2.x
 * @NModuleScope Public
 */
```

Callout	Description
1	The title of the file that holds this script. This annotation is not required, but it can be useful to list here. Any script that loads this module must refer to the name of the file.
2 and 3	The @NApiVersion tag and its value. This tag is not required in a custom module script, but you may want to include it to prevent compatibility problems with scripts that use future versions of SuiteScript. Valid values for this tag are 2.0, 2.x, and 2.X.
4 and 5	The @NModuleScope tag and its value. This tag is optional. You can use it to limit the access of other scripts to this module. For more information about this tag, see Controlling Access to Scripts and Custom Modules .

Entry Point Function

Every custom module script must define an object that is returned when the script’s entry point is invoked. This object could be a static value, such as a string. However, it is far more common for custom module scripts to return a function, as is the case with this script. Because of the way the [Return Statement](#) is set up, the entry point function shown in the following diagram is invoked when the script’s schedule entry point is invoked.

As with an entry point function in a standard entry point script, this function takes a context object as its argument. For more details, see the table that follows the diagram.

```

function scheduleCall ( context ) {
  var newPhoneCall = record.create({
    type: record.Type.PHONE_CALL,
    isDynamic: true
  });

  newPhoneCall.setValue({
    fieldId: 'title',
    value: context.phoneCallTitle
  });

  newPhoneCall.setValue({
    fieldId: 'assigned',
    value: context.phoneCallOwner
  });

  newPhoneCall.setText({
    fieldId: 'phone',
    text: context.phoneNumber
  });

  try {
    var newPhoneCallId = newPhoneCall.save();

    log.debug({
      title: 'Phone call record created successfully',
      details: 'New Phone call record ID: ' + newPhoneCallId
    });
  } catch (e) {
    log.error({
      title: e.name,
      details: e.message
    });
  }
}

```

Callout	Description
1	The context object that is made available when the schedule entry point is invoked. The values of this object's properties are defined in the script that calls this custom module.
2	This statement uses the record.create(options) method to begin the process of creating a phone call record.
3	These statements set fields on the phone call record. They use the properties of the context object, along with the Record.setValue(options) and Record.setText(options) methods. The values of the context object — phoneCallTitle, phoneCallOwner, and phoneCallNumber — must be defined in the script that calls the custom module.
4	This try/catch block attempts to save the new phone call record by using the Record.save(options) method. If the save attempt fails, the block catches and logs the error that caused the problem.

Return Statement

As with an entry point script, the callback function in a custom module script must include a return statement. The return statement must include at least one entry point.

```

return {
  schedule : scheduleCall,
}

```

Callout	Description
1	The custom module's entry point. As with an entry point script, every custom module script must use at least one entry point. The difference is that an entry point script must use a standard entry point that is part of the script type used by the script. A custom module entry point is your creation, so you can use this return statement to give the entry point any name you prefer.
2	A reference to an object. This example references a function. This structure is probably the most common. However, the entry point could reference another object, such as a static value. Whatever object is referenced, it must be defined within the same script file.

Step Four: Upload the Custom Module Script File to NetSuite

After you have created your custom module script file, upload it to your NetSuite File Cabinet.

To upload the script file:

1. In the NetSuite UI, go to Documents > File > SuiteScripts.
2. Click **Add File**.
3. Follow the prompts to locate the phoneCall.js file in your local environment and upload it to the SuiteScripts folder.

Be aware that even after you upload the file, you can edit it from within the File Cabinet, if needed. For details, see the help topic [Editing Files in the File Cabinet](#).

Step Five: Modify the Entry Point Script

After you upload a custom module script file, as described in the last section, you do not need to take any other actions for it to be available. You do not have to create a script record or script deployment record for it. However, for the script to be used, the script must be referenced by a script that has been deployed.

In the next procedure, you modify an entry point script so that it uses the phoneCall.js module. You will update the user event script described in [SuiteScript 2.0 User Event Script Tutorial](#). To update the file, you can use either of the following approaches:

- If you want to copy and paste the updated script directly from this help topic, skip ahead to [Copy the Full Script](#).
- If you want to read about how to make the needed edits yourself, refer to [Update the Script Step by Step](#).

Update the Script Step by Step

This procedure tells you how to update createTask.js so that it uses the phoneCall.js custom module file.

To update the script step by step:

1. Open the createTask.js file for editing.
2. Update the list of dependencies by adding /phoneCall.js to the first argument of the define function.

```
define ( ['N/record', 'N/ui/serverWidget', './phoneCall' ] ,
```

3. Add an additional dependency to the callback function's first argument. This object represents the phone call module. It can be used to access the phone call module's API.

```
function(record, serverWidget, phone) {
```

4. In the script's afterSubmit function, add logic to retrieve the employee's phone number.

```
function myAfterSubmit(context) {

    if (context.type !== context.UserEventType.CREATE)
        return;

    var newEmployeeRecord = context.newRecord;

    var newEmployeeFirstName = newEmployeeRecord.getValue ({
        fieldId: 'firstname'
    });

    var newEmployeeLastName = newEmployeeRecord.getValue ({
        fieldId: 'lastname'
    });

    var newEmployeeSupervisor = newEmployeeRecord.getValue ({
        fieldId: 'supervisor'
    });

    var newEmployeePhoneNumber = newEmployeeRecord.getValue({
        fieldId: 'phone'
    });
}
```

5. Create a block of code that does the following:
 - Create a conditional test that checks to see whether the employee record has values in both the phone number and supervisor fields (Callout 1).
 - Create an object called phoneData that sets values for all fields needed by the phone call module's schedule method (Callout 2).
 - Call the schedule method, passing in the phoneCall object as its argument (Callout 3).

```
1 if (newEmployeeSupervisor && newEmployeePhoneNumber) {

    2 var phoneData =
        {
            3 phoneNumber : newEmployeePhoneNumber, 4
            phoneCallOwner : newEmployeeSupervisor,
            phoneCallTitle : 'Welcome ' + newEmployeeFirstName +
                           ' ' + newEmployeeLastName

        };

    3 phone.schedule(phoneData);

}
```

6. Upload the revised script file to your NetSuite File Cabinet, overwriting the prior version.

Copy the Full Script

The following shows the fully updated user event script. If you haven't already created the script file using the steps described in [Create the Script Step by Step](#), copy and paste the following code into the text editor of your choice. Save the file and name it createTask.js.

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

define ( ['N/record', 'N/ui/serverWidget', './phoneCall' ] ,

function(record, serverWidget, phone ) {

    // In the beforeLoad function, disable the
    // Notes field on the record.

    function myBeforeLoad (context) {

        if (context.type !== context.UserEventType.CREATE)
            return;

        var form = context.form;

        var notesField = form.getField({
            id: 'comments'
        });

        notesField.updateDisplayType({
            displayType: serverWidget.FieldDisplayType.DISABLED
        });
    }

    // In the beforeSubmit function, add a message to the
    // Notes field.

    function myBeforeSubmit(context) {

        if (context.type !== context.UserEventType.CREATE)
            return;

        var newEmployeeRecord = context.newRecord;

        newEmployeeRecord.setValue('comments', 'Orientation date TBD.');
```

```

// In the afterSubmit function, take several actions culminating
// in creating both a task record and a phone call record.

function myAfterSubmit(context) {

    // If the user is not creating a new record, then stop
    // executing.

    if (context.type !== context.UserEventType.CREATE)
        return;

    // Use the context object's newRecord property to retrieve values
    // from the new record.

    var newEmployeeRecord = context.newRecord;

    var newEmployeeFirstName = newEmployeeRecord.getValue ({
        fieldId: 'firstname'
    });

    var newEmployeeLastName = newEmployeeRecord.getValue ({
        fieldId: 'lastname'
    });

    var newEmployeeSupervisor = newEmployeeRecord.getValue ({
        fieldId: 'supervisor'
    });

    var newEmployeePhoneNumber = newEmployeeRecord.getValue({
        fieldId: 'phone'
    });

    // If the user entered a value for the supervisor field,
    // create a task record for the supervisor.

    if (newEmployeeSupervisor) {

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

        newTask.setValue({
            fieldId: 'title',
            value: 'Schedule orientation session for ' +
                newEmployeeFirstName + ' ' + newEmployeeLastName

```

```

    });

    newTask.setValue({
        fieldId: 'assigned',
        value: newEmployeeSupervisor
    });

    try {
        var newTaskId = newTask.save();

        log.debug({
            title: 'Task record created successfully',
            details: 'New task record ID: ' + newTaskId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

}

// If the user entered values for both the supervisor and
// phone number fields, use the phoneCall module to
// schedule a phone call.

if (newEmployeeSupervisor && newEmployeePhoneNumber) {

    var phoneData =

        {
            phoneNumber: newEmployeePhoneNumber,
            phoneCallOwner: newEmployeeSupervisor,
            phoneCallTitle: 'Welcome ' + newEmployeeFirstName +
                ' ' + newEmployeeLastName
        };

    phone.schedule(phoneData);

}

}

return {
    beforeLoad: myBeforeLoad,
    beforeSubmit: myBeforeSubmit,
    afterSubmit: myAfterSubmit
};

```

```
});
```

Step Six: Upload the Revised User Event Script

After you have updated createTask.js, upload it to your NetSuite File Cabinet.

To upload the script file:

1. In the NetSuite UI, go to Documents > File > SuiteScripts.
2. Click **Add File**.
3. Follow the prompts to locate the createTask.js file in your local environment and upload it to the SuiteScripts folder.
4. When the system notifies you that a file with that name already exists, click OK to overwrite the existing file.

Step Seven: Test the Script

Now that the both files have been uploaded, you should verify that the custom module executes as expected.

To test the script:

1. Begin the process of creating a new employee record by selecting Lists > Employees > Employees > New.
2. In the new employee form, check to see whether the Notes field is disabled. If the beforeSubmit entry point function works as expected, the field is gray and cannot be edited.
3. Enter value for required fields. These fields may vary depending on the features enabled in your account and any customizations that exist. Minimally, they include:
 - **Name** — Enter a first name of Susan and last name Johnson.
 - **Subsidiary** — (OneWorld only) Choose a value from the dropdown list.
4. To make sure that the custom module logic is used, enter values in the **Supervisor** and **Phone** fields.
5. Click **Save**. A success message appears, and the system displays the new record in View mode.
6. Verify that the phone call record was created successfully:
 - a. Select Activities > Scheduling > Phone Calls.
 - b. Check the filtering options to make sure that phone calls assigned to all employees are being displayed.
 - c. Verify that a phone call was scheduled to welcome the new employee.



Phone Calls

VIEW

Default

Customize View

New Phone Call

FILTERS









EDIT

EDIT VIEW	INTERNAL ID	SUBJECT	PHONE CALL DATE	PHONE NUMBER	PRIORITY
Edit View	611	Welcome Susan Johnson	11/9/2017	650-555-1212	Medium

Module Dependency Paths

To specify module paths and module names, you can use the [define Object](#) or the [require Function](#).

Avoid using any file extensions in the path.

SuiteScript supports the following:

Path Type	Description
Absolute Paths	Specifies the path from the root folder in the file cabinet. Absolute paths start with a forward slash (/). For example, <code>"/SuiteScripts/MyApp/Util"</code> .
Relative Paths	<p>Specifies the path relative to the dependent module's location in the file cabinet. This provides greater flexibility when moving nested folders containing modules in the file cabinet. Relative Paths start with a period (.).</p> <p>For example, assume that the dependent module is located in the following folder: <code>"/SuiteScripts/MyApp"</code>. The relative path for a sibling file under "SuiteScripts/MyApp" could be <code>"/Util"</code>. The equivalent absolute path in this case would be <code>"/SuiteScripts/MyApp/Util"</code>.</p> <div>  Note: Do not use relative paths in global contexts. For example, when using the <code>require</code> function or SuiteScript Debugger, . </div>
Bundle Virtual Paths	<p>Specifies a bundle path that the NetSuite system can resolve to a file cabinet pointer. A valid bundle ID is required in the bundle virtual path. Bundle virtual paths start with a forward slash followed by a period (/.). For example:</p> <ul style="list-style-type: none"> ▪ <code>/.bundle/<bundle id>/SS2_CustomModuleTest.js/</code> ▪ <code>/.suiteapp/100</code>
Naming a Custom Module	Defines a global identifier used to reference a module by name and not path. Use a require Configuration to set the name.

Absolute Paths

Use an initial forward slash (/) to denote the top-level File Cabinet directory.

In the following example, the Module Loader expects the custom module files `lib1.js` and `lib2.js` to be located in the `SuiteScripts` top-level directory in the File Cabinet.

```
// myModule.js
define(['/SuiteScripts/lib1', '/SuiteScripts/lib2'], // myModule has a dependency on modules lib1 and lib2
...
);
```

Relative Paths

You can specify relative paths to modules in subdirectories from the directory of the current module.

In the following example, the Module Loader expects the custom module files `lib1.js` and `lib2.js` to be located in the `lib` subdirectory, the same File Cabinet directory that contains `myModule.js`.

```
// myModule.js
define(['./lib/lib1', './lib/lib2'], // myModule has a dependency on modules lib1 and lib2
...
);
```

```
);
```

Bundle Virtual Paths

Use a bundle virtual path to point to a SuiteApp file cabinet location. Bundle virtual paths start with a forward slash followed by a period (/.). A valid bundle ID is required in the bundle virtual path.

In the following example, the Module Loader is looking for common libraries in shared bundles.

```
// myModule.js
require(['./suiteapp/100', './bundle/101'], // myModule has a dependency on modules with the bundle id 100 and 101
...
);
```

In certain situations, a virtual path can help when a SuiteApp is moved or the bundle id changes due to deprecation or copying. At the time of copying or deprecation, SuiteScript 2.0 checks the deprecation or copy chain and looks for the file in multiple places. For example, a created bundle is deprecated by a newer version of the bundle, and the newer version is installed in the target account with a different ID. In this case, the scripts using the virtual path of the deprecated bundle in the target account use the new bundle's ID, because the virtual bundle path automatically resolves to the new bundle.

Be aware that the ID for the created bundle in the source account could not be used to specify a virtual bundle path. The virtual bundle path depends on a file path to an existing bundle. At the time the bundle is created in the source account, no such valid file path is created yet.

Naming a Custom Module

Loading a custom module by name

You can set up a custom module name to aid re-use. After you configure a module name, you can require it without knowing the path.

Custom module loading by name also enables better interoperability with third-party libraries and may offer some protection against naming conflicts.

You will need to call `define(id, [dependencies,] moduleObject)` and configure a [require Function](#).

The following steps demonstrate the steps to load a custom module by name:

1. Create your module file. For example `math.js`.
2. Author your custom module name and its contents:
3. To load the module by name, specify the module name (alias) and its path by configuring the `paths` parameter in a JSON file (`myconfig.json`). See [require Configuration](#) for more information.

```
...
{
  "paths": {
    "math": "/SuiteScripts/Example/math"
  }
}
...
```

4. Load the custom module in your SuiteScript 2.0 script by including the `NAmDConfig` tag and passing the name to the [define Object](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType usereventscript
 * @NAmDConfig ./myconfig.json
 */

define(['math'],
  function (math) {
    return {
      beforeLoad: function beforeLoad() {
        log.debug('test', math.add(1,2));
      }
    }
  });
...

```

Custom Module Examples

The following examples demonstrate using the `define` and `require` functions when working with custom modules.

- [Example: Define a custom utility module](#)
- [Example: Import the custom utility module](#)
- [Example: Define a custom module for a SuiteApp](#)
- [Example: Import a third-party JavaScript library](#)

Example: Define a custom utility module

The following file holds the definition of a custom module with a custom API. For example, to allow multiple scripts that depend on an incremental counter utility to require and import this functionality.

To protect against version incompatibility, this script includes the `@NApiVersion` tag.

```
/**
 * counter.js
 * @NApiVersion 2.x
 */
define(function(){
  var counter = 0;

  function incrementValue() {
    counter++;
  }

  function getValue() {
    return counter;
  }

  return {
    increment: incrementValue,
    value: getValue
  }
});

```

```

    }
  });

```

Example: Import the custom utility module

This example uses the [define Object](#) to include a native SuiteScript module and a custom module.

The module's file path is used to pass in the custom utility as a dependency. As a best practice, it does not include the .JS file extension.

```

/**
 * customRecord.js
 * @NApiVersion 2.x
 */
define(['N/record', './counter'],
  function(record, counter){
    function createCustomRec() {
      record.create(...);
      counter.increment();
    }

    return {
      createCustomRecord: createCustomRec
    }
  });

```

Example: Define a custom module for a SuiteApp

To define a bundled custom module that can be exposed to third parties, you must add the `@NModuleScope` JSDoc tag and assign it the value `public`.

```

/**
 * @NApiVersion 2.0
 * @NModuleScope public
 */

```

Next, use the [define Object](#) so that your custom module is recognized as an AMD module. To use a module that is bundled within an external SuiteApp, you need to pass the bundle's file path, containing a valid bundle ID, within the `define()` function as follows:

```

define(['./bundle/<bundle ID>/<module path>'],
  function(<module name>){
    <logic goes here>
  }
);

```

If the bundle ID changes, the module loader will still attempt to locate it. The module loader uses the bundle ID to search the SuiteApp's deprecation path using valid [Bundle Virtual Paths](#).

Example: Import a third-party JavaScript library

- [Import a third-party library](#)
- [Example: Add a non-AMD library](#)

Import a third-party library

Some third-party libraries register as AMD compatible.

In that case, you can specify a require configuration that sets up the path where the module is found. For example, create a JSON configuration file (JsLibraryConfig.json) and save it in the file cabinet:

```
...
{
  "paths": {
    "coolthing": "/SuiteScripts/myFavoriteJsLibrary"
  }
}
...
```

Then, you could use the NAmDConfig JSDoc tag to provide a relative path from the script that needs the coolthing module:

```
/**
 * @NApiVersion 2.x
 * @NScriptType usereventscript
 * @NAmDConfig ./JsLibraryConfig.json
 */

define(['coolthing'],
  function (coolthing)
  {
    return {
      beforeLoad: function beforeLoad(ctx)
      {
        coolthing.times(2, function () {
          log.debug({
            title: 'log',
            details: 'log'
          });
        });
      }
    };
  });
});
```

Similarly, to use a version of jQuery, you can form your `require` configuration as follows:

Add a JSON configuration file (jQueryConfig.json) to the file cabinet and configure the path:

```
...
{
  "paths": {
    "jquery": "/SuiteScripts/myjQueryLib"
  }
}
...
```

From the script using the jQuery module, reference the path to your JSON configuration file in the NAmDConfig tag. For example, `* @NAmDConfig /SuiteScripts/myjQueryLib.json`.

```
...
* @NAmcConfig ./jQueryConfig.json
*/

define(['jquery'], function(jquery) {
});
...
```



Note: If you want to use jQuery with a Suitelet, import it via an on demand client script that is attached to the Suitelet using `Form.clientScriptFileId` or `Form.clientScriptModulePath`.



Important: SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). The NetSuite UI should only be accessed using SuiteScript APIs.

Example: Add a non-AMD library

In this example, the following file would be uploaded to the file cabinet (as a JavaScript file that does not register as AMD compatible).

```
var myMath = {
  add: function(num1, num2) {
    return num1 + num2;
  },
  subtract: function(num1, num2) {
    return num1 - num2;
  },
  multiply: function(num1, num2) {
    return num1 * num2;
  },
  divide: function(num1, num2) {
    return num1 / num2;
  },
}
```

When the library does not register as an AMD module, you need to use a require configuration that specifies the configuration parameters in JSON format. In this case, the `shim` parameter and `paths` parameter are configured in the MathConfig JSON file:

```
...
{
  "paths": {
    "math": "SuiteScripts/Shim Example/math"
  },
  "math": {
    "exports": "myMath"
  }
}
...
```

The module can be loaded as a dependency in an entry point script, such as in a user event script, provided that a valid `@NAmcConfig` tag with a path to the JSON configuration file is included.

```
/**
```

```
* @NApiVersion 2.x
* @NScriptType usereventscript
* @NAmdConfig ./MathConfig.json
*/

define(['math'],
function (math) {
  return {
    beforeLoad: function beforeLoad() {
      log.debug({
        title: 'test',
        details: math.add(1,2)
      });
    }
  }
});
```

Troubleshooting Errors

Module not found

You may see this error if you are using the require function. The require function has no global context. Consequently, relative paths do not work for the require function.

If you receive the Module does not exist error, try replacing any relative paths with an absolute path.

For more information, see [require Function](#).

You do not have permission to load this module.

Review your module scope settings. For a full description of support module scopes, see [Controlling Access to Scripts and Custom Modules](#).

Frequently Asked Questions: Custom Modules

When should I use require versus define?

Always use the [define Object](#) in entry point scripts and when creating new modules. Use the [require Function](#) for loading and using existing modules.

There is a performance advantage to using the require function. Generally, give preference to using the require Function whenever you can. The define Function will imports all dependencies. The require function loads dependencies only as they are needed.

Here is a summary of when to use or not use these functions:

Function	Example	Caveat
define	To define entry point scripts To create new modules. To export modules	If you use a define function, you do not receive the performance benefit of progressive loading.

require	To import modules via relative path To step through code in the SuiteScript debugger To support building a test framework For progressive loading of dependencies	If you use a require function, you may need to import a define that provides correct context.
---------	--	---

Are third-party libraries supported?

Yes. You can define the library as a custom module, set up a global identifier to reference it, and configure it as a dependency. See [Import a third-party library](#) and [Example: Import a third-party JavaScript library](#).

SuiteScript 2.0 Scripting Records and Subrecords

For information on using SuiteScript 2.0 to work with records and subrecords, see the following topics:

- [SuiteScript 2.0 Scripting Records](#)
- [SuiteScript 2.0 Scripting Subrecords](#)

SuiteScript 2.0 Scripting Records

For details on using SuiteScript 2.0 to work with records, see the following sections:

- [SuiteScript 2.0 – Standard and Dynamic Modes](#)
- [SuiteScript 2.0 – Record Modules](#)

SuiteScript 2.0 – Standard and Dynamic Modes

When you create, copy, load, or transform records in SuiteScript, you can work with records in standard or dynamic mode.

Standard mode:

When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in standard mode, the record's body fields and sublist line items are not sourced, calculated, and validated until the record is saved (submitted) with [Record.save\(options\)](#). Standard mode is also called deferred dynamic mode and you see both terms in the SuiteScript help.

Dynamic mode:

When you work with a record in standard mode, in most cases, you don't need to set values in any particular order. After a record is submitted, NetSuite processes the record's body fields and sublist line items in the correct order, regardless of the organization of your script. See [Getting Text in SuiteScript 2.0 Record Modes](#).

When a SuiteScript 2.0 script creates, copies, loads, or transforms a record in dynamic mode, the record's body fields and sublist line items are sourced, calculated, and validated in real-time. A record in dynamic mode emulates the behavior of a record in the UI.

How you can tell if a record is in dynamic mode:

There are two SuiteScript 2.0 properties that indicate if a record is in dynamic mode:

- [Record.isDynamic](#)
- [CurrentRecord.isDynamic](#)

Record Modes and User Event Scripts

Standard mode is always used for user event scripts that instantiate records with the `newRecord` or `oldRecord` object provided by the script context. For that reason, the `SSS_INVALID_API_USAGE` error appears when a user event executes on one of these objects in the following situations:

- When the script executes on a record that is being created, and the script attempts to use [Record.getText\(options\)](#) without first using [Record.setText\(options\)](#) for the same field.

- When the script executes on an existing record or on a record being created through copying, and the script uses [Record.setValue\(options\)](#) on a field before using [Record.getText\(options\)](#) for the same field.

Getting Text in SuiteScript 2.0 Record Modes

In dynamic mode, you can use [Record.getText\(options\)](#) without limitation but, in standard mode, limitations exist. In standard mode, you can use this method **only** in the following cases:

- You can use [Record.getText\(options\)](#) on any field where the script has already used [Record.setText\(options\)](#).
- If you are loading or copying a record, you can use [Record.getText\(options\)](#) on any field except those where the script has already changed the value with [Record.setValue\(options\)](#).

Record Module Method Considerations

Be aware that the [record.create\(options\)](#), [record.copy\(options\)](#), [record.load\(options\)](#), and [record.transform\(options\)](#) methods work in standard mode by default. If you want these methods to work in dynamic mode, you must pass in a specific argument. See the help topic for the applicable method for more information.

SuiteScript 2.0 – Record Modules

With SuiteScript 2.0, you use the [N/record Module](#) and [N/currentRecord Module](#) to script with records.

In **server-side scripts**, use the [N/record Module](#). See the following topics for examples of working with records in server-side scripts:

- [SuiteScript 2.0 User Event Script Sample](#)
- [SuiteScript 2.0 User Event Script Tutorial](#)
- [SuiteScript 2.0 Custom Module Tutorial](#)

In **client scripts**:

- Use [N/currentRecord Module](#) methods to interact with the record that is active in the current client-side context.
- Use the [N/record Module](#) to load and interact with remote records.

See the following topics for examples of working with records in client scripts:

- [SuiteScript Client Script Sample](#)
- [SuiteScript 2.0 Client Scripts on currentRecord](#)
- [SuiteScript 2.0 Remote Objects in Client Scripts](#)
- [Disable Field on Client PageInit using SuiteScript 2.0](#)

SuiteScript 2.0 Scripting Subrecords

For details on using SuiteScript 2.0 to work with subrecords, see the following sections:

- [Understanding Subrecords](#)
- [Subrecord Scripting in SuiteScript 2.0 Compared With 1.0](#)
- [Scripting Subrecords that Occur on Sublist Lines](#)
- [Scripting Subrecords that Occur in Body Fields](#)

Understanding Subrecords

When you use SuiteScript to interact with records, you typically interact with many types of fields. During this process, you may come across fields with a data type of summary. These fields can be populated in only one way: by saving a subrecord to the field. Therefore, if you want to interact with these fields, you must understand how to script with subrecords.

In general, saving subrecords to summary fields is more complex than setting values for other types of fields. However, subrecords are similar to records. So if you know how to work with records, you already know a great deal about working with subrecords. This topic summarizes the similarities and differences.

What Is a Subrecord?

Subrecords represent a way of storing data in NetSuite.

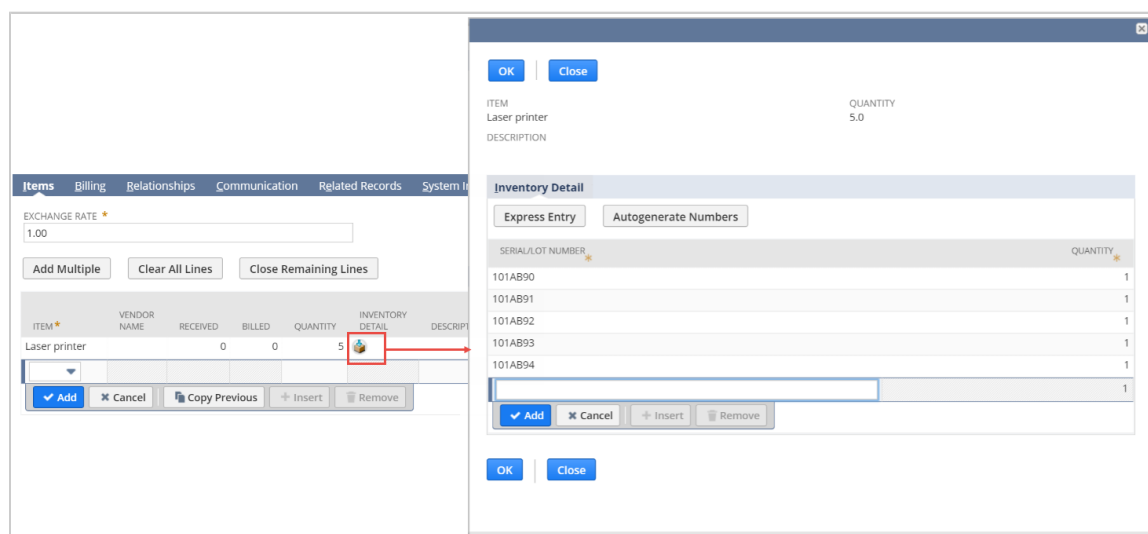
Like records, subrecords are classified by type. Some common types of subrecord include address, inventory detail, and order schedule.

Each subrecord type has a different purpose and includes different fields. For example:

- An address subrecord stores an address. It has fields such as city, state, and zip.
- An order schedule subrecord represents a purchase schedule. It has fields such as startdate and enddate.
- An inventory detail subrecord holds data, such as serial numbers, that describe inventory items. It contains a sublist that holds this data.

At the same time, subrecords differ from records in some ways. For example, while records can exist independently, a subrecord exists solely to hold information about a specific record. You cannot interact with a subrecord outside the context of a parent record.

In the UI, typically you open a subrecord by clicking an icon on the parent record. The subrecord form opens in a separate window. For example, the following illustration shows, at left, a purchase order with one line in its Items sublist. If you click the icon associated with that item, the system opens a window that represents the inventory detail subrecord.



To access this same subrecord from a script, first you would load the purchase order. Then you would use the `Record.getSublistSubrecord(options)` method to open the subrecord. For example:

```

...
// Load the purchase order.

var rec = record.load({
    type: record.Type.PURCHASE_ORDER,
    id: 7,
    isDynamic: false
});

// Retrieve the subrecord. For sublistId, use the ID of the relevant sublist on
// the purchase order record type. For fieldId, use the ID of the summary field
// on the sublist that holds the subrecord.

var subrec = rec.getSublistSubrecord({
    sublistId: 'item',
    line: 0,
    fieldId: 'inventorydetail'
});

...

```

For a full script sample, see [Example: Creating an Inventory Detail Subrecord](#).

Subrecord Scripting Overview

Use the following guidelines when scripting with subrecords:

- [Understand When Subrecords Are Read-Only](#)
- [Look Up Details About the Summary Field](#)
- [Look Up the Subrecord's Field and Sublist IDs](#)
- [Use Record Methods to Get and Set Values](#)
- [Do Not Explicitly Save a Subrecord](#)

Understand When Subrecords Are Read-Only

You can work with subrecords in both client and server-side scripts. However, subrecords are read-only when their parent records are retrieved in either of the following ways:

- Through the context object provided to a client script, or through `currentRecord.get()`
- Through the context object provided to a `beforeLoad` user event script

For more details, see [Supported Deployments for Subrecord Scripting](#).

Look Up Details About the Summary Field

Before you can script with a subrecord, you must have some knowledge of the summary field that holds the subrecord. In addition to the summary field's ID, you must know whether the summary field is situated on the body of the parent record or on one of its sublists. You need both pieces of information to instantiate the subrecord. If the summary field is a sublist field, you also need the relevant sublist ID.

If you are not sure where the field is situated, you can review the record in the UI. You can also check the reference page for the record in the [SuiteScript Records Browser](#). On the reference page for each

record type, the **Fields** table lists all of the record type's body fields. The tables listed under the heading **Sublists** show sublist fields. For more details, see [Finding Details About Parent Record Types](#).

Look Up the Subrecord's Field and Sublist IDs

Like records, all subrecords have required and optional fields. To set values for these fields, you must have the field IDs. To work with a subrecord's sublist fields, you must also have the sublist ID. You can find both types of information in the [SuiteScript Records Browser](#). Each subrecord type is listed in the browser alongside the available record types. For each subrecord type, a reference page includes the IDs for all of the elements on the subrecord, including a sublist, if one exists, and all of the subrecord's fields. For more details, see [Finding Details About Subrecord Types](#).

Additionally, if you have the Show Internal IDs preference enabled, you can use the UI to find the IDs for subrecord body fields. To view a field's ID, click its label. In response, the system displays a popup window that shows the ID. For help enabling this preference, see the help topic [Setting the Internal ID Preference](#).

Use Record Methods to Get and Set Values

When you script with a subrecord, many aspects of the scripting process are identical to the process of scripting with records.

For example, when your script instantiates a subrecord, the system returns one of the same objects that it uses to represent records. These objects include `record.Record` and `currentRecord.CurrentRecord`.

Additionally, you use many of the same methods to interact with subrecords as you do with records. For example:

- In a server-side script, you use `Record.setValue(options)` to set a value on either a record or subrecord body field.
- In a client script, you can use `CurrentRecord.getValue(options)` to retrieve the value stored in either a record or subrecord body field.

A small number of record methods and properties are unavailable to subrecords. These exceptions are noted in the property descriptions in the [N/record Module](#) and [N/currentRecord Module](#) topics.

Do Not Explicitly Save a Subrecord

After you have created or updated a subrecord, you do not explicitly save it. Rather, after you have set all required fields on the subrecord, you simply save the parent record. When you save the record, the subrecord is also saved.

Example: Creating an Inventory Detail Subrecord

The following example shows a typical approach to creating a subrecord.

This script creates a purchase order with one inventory item in its sublist. The sample also creates an inventory detail subrecord to store a lot number for the item.

Before using this sample in your NetSuite account, do the following:

- Make sure the [Advanced Bin / Numbered Inventory Management](#) feature is enabled, at Setup > Company > Enable Features, on the Items & Inventory subtab.
- Verify that you have at least one vendor, one location, and one lot-numbered inventory item defined in your system. Make a note of each record's internal ID.
- Where noted in the script sample comments, replace the hardcoded IDs with valid values from your NetSuite account.

This example uses standard mode, but you could also add the subrecord using dynamic mode. For more details about both approaches, see [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#).

Note: For help deploying a user event script, see [SuiteScript 2.0 Entry Point Script Creation and Deployment](#).

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

define(['N/record'],

function(record) {

    function myAfterSubmit(context) {

        // Create the purchase order.

        var rec = record.create({
            type: record.Type.PURCHASE_ORDER,
            isDynamic: false
        });

        // Set body fields on the purchase order. Replace both of these
        // hardcoded values with valid values from your NetSuite account.

        rec.setValue({
            fieldId: 'entity',
            value: '2'
        });

        rec.setValue({
            fieldId: 'location',
            value: '2'
        });

        // Insert a line in the item sublist.

        rec.insertLine({
            sublistId: 'item',
            line: 0
        });

        // Set the required fields on the line. Replace the hardcoded value
        // for the item field with a valid value from your NetSuite account.

        rec.setSublistValue({
            sublistId: 'item',
            fieldId: 'item',
            line: 0,
```

```

        value: '7'
    });

    rec.setSublistValue({
        sublistId: 'item',
        fieldId: 'quantity',
        line: 0,
        value: 1
    });

    // Instantiate the subrecord. To use this method, you must
    // provide the ID of the sublist, the number of the line you want
    // to interact with, and the ID of the summary field.

    var subrec = rec.getSublistSubrecord({
        sublistId: 'item',
        line: 0,
        fieldId: 'inventorydetail'
    });

    // Insert a line in the subrecord's inventory assignment sublist.

    subrec.insertLine({
        sublistId: 'inventoryassignment',
        line: 0
    });

    subrec.setSublistValue({
        sublistId: 'inventoryassignment',
        fieldId: 'quantity',
        line: 0,
        value: 1
    });

    // Set the lot number for the item. Although this value is
    // hardcoded, you do not have to change it, because it doesn't
    // reference a record in your account. For the purpose of this
    // sample, the value can be any string.

    subrec.setSublistValue({
        sublistId: 'inventoryassignment',
        fieldId: 'receiptinventorynumber',
        line: 0,
        value: '01234'
    });

    // Save the record. Note that the subrecord object does
    // not have to be explicitly saved.

    try {

```

```

    var recId = rec.save();

    log.debug({
        title: 'Record created successfully',
        details: 'Id: ' + recId
    });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }
}

return {
    afterSubmit: myAfterSubmit
};
});

```

For additional samples, see [Scripting Subrecords that Occur on Sublist Lines](#) and [Scripting Subrecords that Occur in Body Fields](#).

Supported Deployments for Subrecord Scripting

A script can interact with subrecord instances only if it uses a supported deployment.

To understand supported deployments, it's important to understand that every subrecord has a parent record. For more details on this relationship, see [Understanding Subrecords](#).

The following types of deployments are supported:

- [Server-side Scripts Deployed on Parent Records](#)
- [Client Scripts Deployed on Parent Records \(with Limitations\)](#)
- [Client Scripts Deployed on Custom Address Forms](#)

Server-side Scripts Deployed on Parent Records

If you want to create a server-side script that interacts with a subrecord, you can deploy the script on the parent record type. Alternatively, you can deploy the script on a different record type — but then use the script to load the parent record. After loading the parent record, you can interact with the subrecord in the context of its parent.

Subrecord methods are not supported in beforeLoad user event scripts, except if the script creates or loads another record that is a parent, and interacts with the subrecord in the context of that parent.

You cannot deploy a server-side script directly to a subrecord type.

Client Scripts Deployed on Parent Records (with Limitations)

A client script may not create subrecords on the current record and is limited to read-only access of existing subrecords on the current record. The client script may remove the subrecord from the current record.

You cannot deploy a client script directly on a subrecord type. However, you can customize an address form, as described in the following section.

You cannot deploy a client script directly to a subrecord type.

Client Scripts Deployed on Custom Address Forms

If appropriate, you can create custom forms for the address subrecord. This process is described in [Customizing Address Forms](#). When working with a custom address form, you can attach a client script to the form with the Custom Code subtab. In these types of scripts, you can interact with the subrecord using the same methods as you would with a record. This process is not covered in this chapter.

Body Field Subrecords and Sublist Subrecords

When a subrecord occurs on a record, it is always represented by a single field on the record. In the [SuiteScript Records Browser](#), this field is always listed as a field of type summary.

A field that contains a subrecord can exist either as a body or sublist field. For example, the subsidiary record has a body field called mainaddress, which stores an address subrecord. By contrast, the employee record permits the creation of multiple addresses, and each address is described in a sublist line. The sublist includes a field that contains the address subrecord instance.

This difference in where the summary field is placed affects the way you instantiate the subrecord.

For an example of each type of placement, see the following sections:

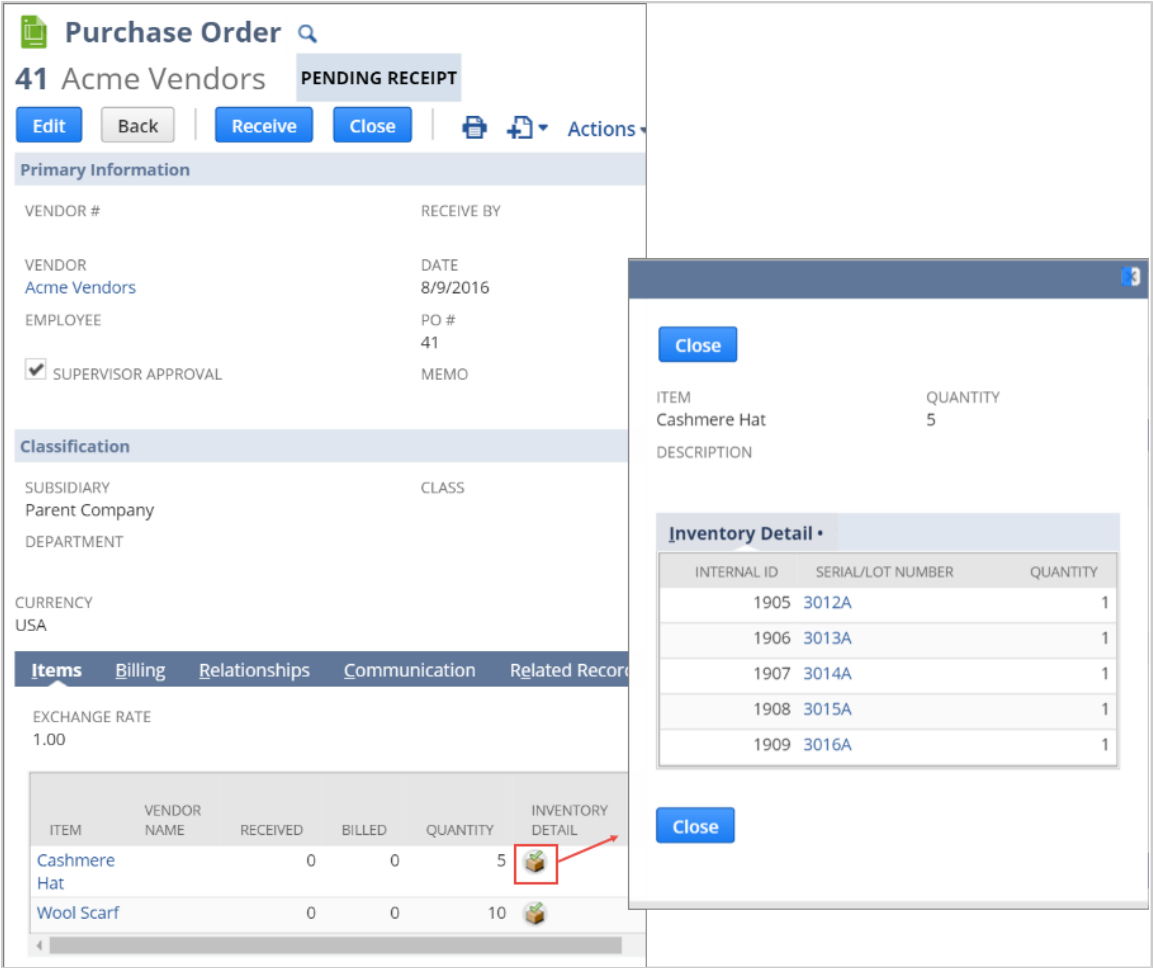
- [Example: Subrecord that Occurs in a Sublist Field](#)
- [Example: Subrecord that Occurs in a Body Field](#)

Example: Subrecord that Occurs in a Sublist Field

Many subrecord types can occur in a sublist field.

For example, depending on the features enabled in your account, the item sublist of a purchase order record can include an Inventory Detail column. If the item on the line is a serialized or lot-numbered item, you can create a subrecord instance in this column.

In the UI, you can view and set values in this subrecord by clicking the icon in the Inventory Detail column. Clicking this icon opens a new window that represents the subrecord.



Be aware that other fields on the sublist line are not part of the subrecord. For example, in the preceding screenshot, the values in the Item, Vendor Name, Received, Billed, and Quantity columns are not part of the subrecord.

For details on using SuiteScript 2.0 to work with subrecords that exist on sublist lines, see [Scripting Subrecords that Occur on Sublist Lines](#).

Example: Subrecord that Occurs in a Body Field

A subrecord can also occur in a body field. In fact, the same type of subrecord that appears as a sublist field on one record type can appear as a body field on another record type.

For example, depending on the configuration of your account, the assembly build record can include an Inventory Detail body field. If the item in the record's Assembly field is a serialized or lot-numbered inventory item, you can create an Inventory Detail subrecord in the Inventory Detail field.

In the UI, you can view and set values in this subrecord by clicking the icon under the Inventory Detail label and opening a new window.

The screenshot shows the 'Assembly Build' interface. On the left, the 'Primary Information' section contains fields for 'REFERENCE #', 'QUANTITY TO BUILD' (set to 3), 'ASSEMBLY' (set to 'Winter gift set'), 'PROJECTED VALUE' (set to 0.00), 'REVISION', and 'BUILDABLE QUANTITY' (set to 0). A red box highlights the 'INVENTORY DETAIL' subrecord icon, with a red arrow pointing to a detailed view on the right. The detailed view shows the 'Inventory Detail' table with columns 'INTERNAL ID', 'SERIAL/LOT NUMBER', and 'QUANTITY'. It lists three items: 1603 012345A, 1903 012345B, and 1904 012345C, each with a quantity of 1. The interface also includes 'Save', 'Cancel', and 'Reset' buttons at the top left, and 'Close' buttons at the top and bottom of the detailed view.

INTERNAL ID	SERIAL/LOT NUMBER	QUANTITY
1603	012345A	1
1903	012345B	1
1904	012345C	1

For details on using SuiteScript 2.0 to work with subrecords that exist in body fields, see [Scripting Subrecords that Occur in Body Fields](#).

Structure of a Subrecord

The fields available in a subrecord vary depending on the subrecord's type. A subrecord can have body fields, a sublist, or both.

When working with a subrecord's fields, be aware of the following:

- After you instantiate a subrecord, you can set values on the subrecord's body and sublist fields with the same methods as you would on a record.
- In general, the sublists that exist within subrecords are not labeled in the UI. To find the name of a subrecord's sublist, refer to the [SuiteScript Records Browser](#). For details on using the Records Browser to find details on subrecords, see [Finding Subrecord Details in the Records Browser](#).

For examples of subrecords that are structured in different ways, see the following sections:

- [Example: Writable Body Fields](#)
- [Example: Writable Sublist](#)
- [Example: Writable Body Fields and Sublist](#)

Example: Writable Body Fields

The address subrecord has several writable body fields, such as city, state, and zip. It has no sublist.

After you instantiate an address subrecord, you can set values for its body fields with the [Record.setValue\(options\)](#) method, the same as if you were setting values on an instance of a record. For more details, see [Example: Creating an Address Sublist Subrecord](#) and [Example: Creating an Address on a Subsidiary Record](#).

Example: Writable Sublist

The landed cost subrecord has a sublist that lets you list individual expenses associated with merchandise you have received. The body fields on this subrecord are read-only, but the sublist is writable. To add details about an expense, you add a line to the sublist.

OK | Close

ITEM: Peacoat TOTAL: 345.00

Add Multiple | Clear All Lines

COST CATEGORY *	AMOUNT *
Customs	115.00
Shipping and Handling	100.00
Port Fees	130.00

▼

✓ Add | ✕ Cancel | + Insert | 🗑 Remove

OK | Close

After you instantiate a landed cost subrecord, you can set values for its sublist fields with the same methods you would to set values on a record's sublist: `setSublistValue()` and `setCurrentSublistValue()`.

For more details, see [Example: Creating a Landed Cost Sublist Subrecord](#).

Example: Writable Body Fields and Sublist

The order schedule subrecord has a sublist that lets you configure how and when upcoming purchase orders are to be created. This subrecord has body fields that let you specify various qualities of the schedule, such as whether individual purchase orders must be created manually. It also has a sublist that lets you enter dates for the upcoming purchase orders.

CREATE PURCHASE ORDERS *

Manually

CREATE SCHEDULE *

Manually

TOTAL QUANTITY
40

Clear All Lines Autogenerate

RELEASE	ORDER	DATE *	QUANTITY *	MEMO
		9/15/2016	10	
		1/16/2017	10	
		4/17/2017	10	
		7/17/2017	10	

✓ Add ✕ Cancel + Insert 🗑 Remove

OK Close

Again, after you instantiate an order schedule subrecord, you can set values its fields with the same methods you would use to set values on a record. For body fields, use `setValue()`. For sublist fields, use `setCurrentSublistValue()` or `setSublistValue()`.

For more details, see [Example: Creating an Order Schedule Sublist Subrecord](#).

Finding Subrecord Details in the Records Browser

As with records, you can find important information about subrecords in the [SuiteScript Records Browser](#). The browser includes details about each subrecord type. It also includes information about record types that can be parents to subrecords. For more guidance, see the following sections:

- [Finding Details About Parent Record Types](#)
- [Finding Details About Subrecord Types](#)

Finding Details About Parent Record Types

Every subrecord instance must have a parent record. An instance of a subrecord exists only to provide information about an instance of a record.

The parent record includes a field that contains, or references, the subrecord. To create a subrecord instance on a record, you must reference this field. These fields are always identified in the Records Browser as fields of type summary.

item - Item

Internal ID	Type	Label	Required
amount	currency	Amount	false
billvariancestatus	text		false
catchupperperiod	select		false
class	select	Class	false
deferrevrec	checkbox		false
department	select	Department	false
description	textarea	Description	false
id	text		false
isvsoebundle	text		false
item	select	Item	true
itemsubtype	text		false
itemtype	text		false
line	text		false
linenumber	integer		false
location	select	Location	false
matrixtype	text		false
options	text		false
orderschedule	summary	Schedule	false
quantity	float	Quantity	false
quantityordered	float	Quantity Ordered	false
rate	rate	Rate	false
rateschedule	text		false
units	select	Units	false
vendorname	text	Vendor Name	false

Summary fields can occur either on the body of the parent record or in a sublist. It is important to know where the field occurs, because this distinction affects how you instantiate the subrecord. In the reference page for each record type in the Records Browser, the **Fields** table lists all of the record type's body fields. The tables listed under the heading **Sublists** shows sublist fields. See also [Body Field Subrecords and Sublist Subrecords](#).

In some cases, the values set for other fields on a record can affect the availability or behavior of the summary field. For example:

- On an assembly build record, the availability of the inventorydetail summary field varies depending on the value of the record's item field. The summary field is available only if the item field references a serialized or lot-numbered assembly item. For an example of working with this record-subrecord combination, see [Example: Creating an Inventory Detail Subrecord on a Body Field](#).
- On a vendor bill record, the landedcost summary field is available only if the landedcostperline body field is set to true. For an example of working with this record-subrecord combination, see [Example: Creating a Landed Cost Sublist Subrecord](#).
- On a sales transaction, the value of the shippingaddress summary field is affected by the shipaddresslist field. For general details about shipping and billing addresses, see [Scripting Transaction Shipping and Billing Addresses](#). For a script sample, see [Example: Using SuiteScript 2.0 to Create a New Shipping Address](#).

Finding Details About Subrecord Types

As with record types, the Records Browser includes a reference page for each subrecord type. Subrecords are listed alphabetically with records.

Each subrecord is identified by a label displayed beneath the internal ID.

Order Schedule

Internal ID: orderschedule

Subrecord

Fields

Internal ID	Type	nlapiSubmitField	Label	Required	
createpurchaseorder	select	false	Create Purchase Orders	true	
createschedule	select	false	Create Schedule	true	
currencyprecision	integer	false		false	
enddate	date	false	End Date	false	
externalid	text	false	ExternalId	false	
item	integer	false		false	
releasefrequency	select	false	Release Frequency	false	
startdate	date	false	Start Date	false	
total	float	false	Total Quantity	false	

Sublists

schedule - Schedule

Internal ID	Type	Label	Required
amount	poscurrency		false
id	integer		false
memo	text	Memo	false
orderschedule	integer		false
purchaseorder	select	Order	false
quantity	posfloat	Quantity	true
release	checkbox	Release	false
trandate	date	Date	true

As with record types, the reference page shows the subrecord's scriptable fields and sublists.

Note that in the UI, the sublists of subrecords typically are not labeled. However, the Records Browser displays the name of each sublist.

Understanding the Address Subrecord

The address subrecord has certain qualities that are unique. These characteristics can make the process of interacting with the address subrecord different from other subrecords. For details, see the following sections:

- [Billing and Shipping Addresses Can Be Sourced from Other Records](#)
- [The Address Subrecord Can Have Custom Forms](#)
- [Address Data Is Summarized on the Parent Record](#)



Note: See also [Scripting Transaction Shipping and Billing Addresses](#).

Billing and Shipping Addresses Can Be Sourced from Other Records

With most subrecord types, an instance of the subrecord is unique to the record where it was created. However, in some cases, a single address subrecord instance can be referenced by multiple records.

For example: You can create multiple addresses for an entity. If you later create a transaction for that entity, you can use one of the addresses defined on the entity record as the shipping or billing address for the transaction. For this reason, setting a value for a shipping or billing address is in some cases slightly different from the way that you set other address summary fields. For details, see [Addresses Can Be Sourced from Entity Records](#).

The Address Subrecord Can Have Custom Forms

Compared with other subrecords, the address subrecord is unique in that you can create custom entry forms for it. For example, you may want to create custom forms for different countries.

If your account has multiple address forms, and if you are not seeing the expected results, the reason could be related to the form. In particular, if your script is using dynamic mode, the first value you set on the subrecord form should be country. The reason is that if you set a value for country that differs from the default, the form resets when the country value changes. Therefore, as a best practice, set the country value first.

Address Data Is Summarized on the Parent Record

With most types of subrecords, details that you enter on the subrecord are not summarized on the parent record when you view it in the UI. You have to open the subrecord in its own window to view its data.

The address subrecord is an exception to this rule. After you enter details into an address subrecord and return to the main view of the record, typically the system displays a summary of the details you entered. For example, in the following screenshot, subrecord data is summarized in the Address column.

Relationships								Communication		Address		Sales		Marketing		Support		Financial		Preferences		System Information		
ID		DEFAULT SHIPPING		DEFAULT BILLING		RESIDENTIAL ADDRESS		LABEL		ADDRESS		EDIT												
2359		Yes		Yes				Company Headquarters		2955 Campus Drive Suite 100 San Mateo CA 94403-2511 United States														

This summary represents the value of one field on the address subrecord called `addrtext`. This value is created through the use of a template and generated from other values entered on the subrecord, such as the values for the city and state fields. Each address form can have its own template for determining the `addrtext` value. You can view the template for any address form by viewing its record at Customization > Forms > Address Forms.

An error in the `addrtext` field does not necessarily signify an error in the other values saved to the subrecord. If an error exists in the summary, review the template to make sure that it is capturing the values you intend.

To view the values for all of the fields on the subrecord, do one of the following:

- Use one of the following methods to retrieve the subrecord: `getCurrentSublistSubrecord()`, `getSublistSubrecord()`, or `getSubrecord()`. For an example, see [Example: Retrieving an Address Subrecord](#).
- In the UI, open the subrecord for editing in its own window.

Client Scripts Attached to the Address Subrecord

Client scripts attached to address subrecords on transaction records may execute on the server side in addition to the client side. Be aware that this is expected behavior. For logic in a client script attached

to an address subrecord to execute only once, wrap the logic in an `if` statement that immediately exits the script on the server side. For example:

```
if (typeof document !== 'undefined') {
  // client script logic
}
```

If a client script attached to an address subrecord loads the [N/currentRecord Module](#), the script fails. Although you cannot load the `N/currentRecord` module in a client script that is attached to the address subrecord, you can obtain `currentRecord` from `context.currentRecord`:

```
define([],
  function() {
    function saveRecord(context) {
      var currentRecord = context.currentRecord;
      return true;
    }
    return {
      saveRecord: saveRecord
    }
  }
);
```

See [currentRecord.CurrentRecord](#).

You can attach client scripts to custom entry forms, custom transaction forms, or custom address forms. See [Attaching a Client Script to a Form](#).

For information about client scripts, see [SuiteScript 2.0 Client Script Type](#).

Subrecord Scripting in SuiteScript 2.0 Compared With 1.0

Compared with SuiteScript 1.0, SuiteScript 2.0 introduces the following changes in how you script subrecords:

- [A Single Method Lets You Create and Load Subrecords](#)
- [Subrecords Do Not Have to Be Explicitly Saved](#)
- [To Create Addresses, You Must Use Subrecord Methods](#)

A Single Method Lets You Create and Load Subrecords

In SuiteScript 1.0, you use one set of APIs to create subrecords and another set to edit subrecords. For example, you could use `nlapiCreateSubrecord` to create a subrecord. You could use `nlapiEditSubrecord` to edit a subrecord.

By contrast, in SuiteScript 2.0, any method that creates a subrecord can also be used to load that subrecord for the purpose of editing it. These methods all have the word **get** in their names. For example, you use the `getSubrecord()` method to create or load a subrecord that exists on the body of a record. You use the `getSublistSubrecord()` or `getCurrentSublistSubrecord()` method to create or load a subrecord that exists on a sublist line.

When you use any of these methods, the system responds with the following logic:

- If a subrecord instance already exists in the specified field, the subrecord is loaded.
- If no subrecord instance exists, the system creates one. You can then set values on the field. The subrecord is saved when you save the record.

Subrecords Do Not Have to Be Explicitly Saved

In SuiteScript 1.0, you had to explicitly save a subrecord prior to saving the record. However, in SuiteScript 2.0, after you create a subrecord or make changes to one, you are not required to explicitly save the subrecord (and no methods exist for that purpose). Your new subrecord is saved at the time you save the record. The same rule applies if you make changes to an existing subrecord. Your updates are saved at the time you save the record.

To Create Addresses, You Must Use Subrecord Methods

The address subrecord was introduced in version 2014.2. Prior to that time, each address was represented on a record as a series of body fields or as a line in a sublist.

After the introduction of the address subrecord, SuiteScript 1.0 was enhanced to support two methods of interacting with addresses: In 1.0, you have the choice of interacting with addresses using subrecord APIs, which is the preferred method. But in 1.0 you can also interact with addresses using the legacy approach: setting values for the address body and sublist fields that used to exist. This support was made possible by logic added to the system that read the values set in this manner and created an address subrecord on behalf of the 1.0 script. Because this support exists in 1.0, these deprecated fields are displayed in the SuiteScript Records Browser as available fields.

However, in SuiteScript 2.0, to create an address, you **must** use subrecord methods. The system does not provide logic for the legacy address body and sublist fields. For that reason, to create, edit, or load an address in SuiteScript 2.0, you must instantiate the address subrecord by referencing the appropriate summary field.

Scripting Subrecords that Occur on Sublist Lines

In many cases, a subrecord is accessed through a field on a sublist line. For example:

- Every purchase order must include a list of items. Depending on the configuration of each item, the sublist line may be required to include an inventory detail subrecord.
- A vendor bill may include a list of items. If the bill is configured to track landed cost per line, each line in the Items sublist may include a landed cost subrecord.
- An employee may have multiple addresses. Each address is stored in a subrecord, and each subrecord is associated with a line in the Address sublist.

In each case, the sublist line also has fields that are not part of the subrecord. The subrecord itself is associated with only one field on the sublist line. In the SuiteScript Records Browser, this field is always identified as a field of type summary.

For more details, see the following sections:

- [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#)
- [Using SuiteScript 2.0 to Edit a Subrecord that Occurs in a Sublist Field](#)
- [Using SuiteScript 2.0 to Retrieve a Sublist Subrecord](#)

Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field

Depending on the record type and other variables, a line on a record's sublist can include a field that references a subrecord. In many cases, you must add the subrecord at the time you are creating the sublist line. In other cases, you can go back and add the subrecord later.

To create a sublist subrecord, your script must use the [N/record Module](#). The script can use either dynamic or standard mode. For details, see the following sections:

- [Creating a Sublist Subrecord in Dynamic Mode](#)
- [Creating a Sublist Subrecord in Standard Mode](#)

Subrecords can also occur in the body field of a record. For details on working with subrecords when occur in body fields, see [Scripting Subrecords that Occur in Body Fields](#). For an overview of the difference between these two types of placement, see [Body Field Subrecords and Sublist Subrecords](#).

 **Note:** For more details about the methods referenced in this topic, see [Record Object Members](#).

Creating a Sublist Subrecord in Dynamic Mode

If your script uses dynamic mode, you can use the following procedure to create a subrecord in a sublist field.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#).

To create a sublist subrecord in dynamic mode:

1. If you want to add a new record, create it and set the required body fields. If you want to update an existing record, load the record.
2. Do one of the following:
 - If you want to create a new sublist line, create the line using the [Record.selectNewLine\(options\)](#) method. Set any required values on the sublist line with the [Record.setCurrentSublistValue\(options\)](#) method.
 - If you want to add a new subrecord to an existing sublist line, identify that line using the [Record.selectLine\(options\)](#) method.
3. Create the new subrecord with the [Record.getCurrentSublistSubrecord\(options\)](#) method. This method takes two arguments:
 - A `sublistId`, which identifies the sublist.
 - A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.

For example, you could use an expression like the following to create an order schedule subrecord on an item sublist:

```
...
var orderScheduleSubrecord = blanketPurchaseOrder.getCurrentSublistSubrecord({
  sublistId: 'item',
  fieldId: 'orderschedule'
});
...
```

4. As appropriate, set body fields on the subrecord with the [Record.setValue\(options\)](#) method. For example, on an order schedule subrecord, you could use the following expression to set a value for the Create Purchase Orders select field.

```
...
orderScheduleSubrecord.setValue({
  fieldId: 'createpurchaseorder',
  value: 'LEAD'
```

```
});
...
```

Be aware that not all subrecords have writable body fields.

5. If the subrecord has a sublist, generally you are required to add at least one line to the sublist. For each line, use the following guidelines:
 - Create the line with the [Record.selectNewLine\(options\)](#) method.
 - Set required values on the line with the [Record.setCurrentSublistValue\(options\)](#) method.
 - Save the subrecord's sublist line with the [Record.commitLine\(options\)](#) method.

For example, if you create an order schedule subrecord, you could use the following expressions to create a line on the subrecord's schedule sublist:

```
...
orderScheduleSubrecord.selectNewLine
    sublistId: 'schedule',
});

orderScheduleSubrecord.setCurrentSublistValue({
    sublistId: 'schedule',
    fieldId: 'quantity',
    value: 1
});

orderScheduleSubrecord.setCurrentSublistValue({
    sublistId: 'schedule',
    fieldId: 'trandate',
    value: dateVariable
});

orderScheduleSubrecord.commitLine({
    sublistId: 'schedule'
});
...
```

6. Save the sublist line that holds the subrecord with the [Record.commitLine\(options\)](#) method.
7. Save the record with the [Record.save\(options\)](#) method.

Note: For a full script sample, see [Example: Creating an Inventory Detail Sublist Subrecord](#) and [Example: Creating an Address Sublist Subrecord](#).

Creating a Sublist Subrecord in Standard Mode

If your script uses standard mode, you can use the following procedure to create a subrecord in a sublist field.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To create a sublist subrecord in standard mode:

1. If you want to add a new record, create it and set the required body fields. If you want to update an existing record, load the record.
2. If you want to create a new sublist line, create the line with the [Record.insertLine\(options\)](#) method. Set any required fields on the sublist line.

3. Create the new subrecord with the [Record.getSublistSubrecord\(options\)](#) method. This method takes three arguments:
 - A sublistId, which identifies the sublist.
 - A fieldId, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.
 - A line number.

For example, you could use an expression like the following to create an inventory detail subrecord on the first line in an item sublist:

```
...
inventoryDetailSubrecord = rec.getSublistSubrecord({
  sublistId: 'item',
  fieldId: 'inventorydetail',
  line: 0
});
...
```

4. Set body fields on the subrecord with the [Record.setValue\(options\)](#) method. Be aware that not all subrecords have writable body fields.
5. If the subrecord has a sublist, generally you are required to add at least one line to the sublist. For each line, use the following guidelines:
 - Use the [Record.insertLine\(options\)](#) method to create the line.
 - Set values on the line with the [Record.setSublistValue\(options\)](#) method.

For example, if you were creating an order schedule subrecord, you could use the following expressions to create a line on the subrecord's schedule sublist:

```
...
subrecordInvDetail.setSublistValue({
  sublistId: 'inventoryassignment',
  fieldId: 'receiptinventorynumber',
  line: 0,
  value: '012345'
});
...
```

6. Save the record with the [Record.save\(options\)](#) method.

 **Note:** For a full script sample of creating a sublist subrecord in standard mode, see [Example: Creating a Landed Cost Sublist Subrecord](#).

Example: Creating an Inventory Detail Sublist Subrecord

The following example shows how to create a purchase order record that includes an inventory detail subrecord. The script adds one line to the item sublist and creates an inventory detail subrecord on that line.

To use this sample, you must meet the following prerequisites:

- The Advanced Bin / Numbered Inventory Management feature must be enabled at Setup > Company > Enable Features , on the Items & Inventory subtab.
- The item you add to the sublist should be a lot-numbered inventory item.
- The receiptinventorynumber value must be unique in your system.

This example uses dynamic mode, but you could also add the subrecord using standard mode. For general details about using either approach to add a sublist subrecord, see [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the purchase order.

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

        // Set body fields on the purchase order.

        rec.setValue({
            fieldId: 'entity',
            value: '1663'
        });

        rec.setValue({
            fieldId: 'location',
            value: '6'
        });

        // Create one line in the item sublist.

        rec.selectNewLine({
            sublistId: 'item'
        });

        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'item',
            value: '299'
        });

        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'quantity',
            value: 1
        });
    }
});
```

```

// Create the subrecord for that line.

var subrec = rec.getCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'inventorydetail'
});

// Add a line to the subrecord's inventory assignment sublist.

subrec.selectNewLine({
    sublistId: 'inventoryassignment'
});

subrec.setCurrentSublistValue({
    sublistId: 'inventoryassignment',
    fieldId: 'quantity',
    value: 2
});

subrec.setCurrentSublistValue({
    sublistId: 'inventoryassignment',
    fieldId: 'receiptinventorynumber',
    value: '01234'
});

// Save the line in the subrecord's sublist.

subrec.commitLine({
    sublistId: 'inventoryassignment'
});

// Save the line in the record's sublist

rec.commitLine({
    sublistId: 'item'
});

// Save the record.

try {

    var recId = rec.save();

    log.debug({
        title: 'Record created successfully',
        details: 'Id: ' + recId
    });

} catch (e) {

```



```

        log.error({
            title: e.name,
            details: e.message
        });
    }
}

return {
    afterSubmit: afterSubmit
};
});

```

Example: Creating an Address Sublist Subrecord

The following example shows how to create an employee record and populate the Address sublist with one line. The script also creates an address subrecord on the sublist line.

This example uses dynamic mode, but you could also add the subrecord using standard mode. For general details about using either approach to add a sublist subrecord, see [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the record.

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

        // Set the required body fields.

        rec.setValue({
            fieldId: 'firstname',
            value: 'John'
        });

        rec.setValue({
            fieldId: 'lastname',
            value: 'Smith'
        });
    }
}

```

```

rec.setValue({
    fieldId: 'subsidiary',
    value: '1'
});

// Create a line in the Address sublist.

rec.selectNewLine({
    sublistId: 'addressbook'
});

// Set an optional field on the sublist line.

rec.setCurrentSublistValue({
    sublistId: 'addressbook',
    fieldId: 'label',
    value: 'Primary Address'
});

// Create an address subrecord for the line.

var subrec = rec.getCurrentSublistSubrecord({
    sublistId: 'addressbook',
    fieldId: 'addressbookaddress'
});

// Set body fields on the subrecord. Because the script uses
// dynamic mode, you should set the country value first. The country
// value determines which address form is to be used, so by setting
// this value first, you ensure that the values for the rest
// of the form's fields will be set properly.

subrec.setValue({
    fieldId: 'country',
    value: 'US'
});

subrec.setValue({
    fieldId: 'city',
    value: 'San Mateo'
});

subrec.setValue({
    fieldId: 'state',
    value: 'CA'
});

subrec.setValue({
    fieldId: 'zip',
    value: '94403'
});

```

```

    subrec.setValue({
        fieldId: 'addr1',
        value: '2955 Campus Drive'
    });

    subrec.setValue({
        fieldId: 'addr2',
        value: 'Suite 100'
    });

    // Save the sublist line.

    rec.commitLine({
        sublistId: 'addressbook'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });

    }

    return {
        afterSubmit: afterSubmit
    };
});

```

Example: Creating an Order Schedule Sublist Subrecord

The following example creates a blanket purchase order record. It creates one line in the item sublist and creates an order schedule subrecord on that line.

To use this example, the Blanket Purchase Order feature must be enabled at Setup > Company > Enable Features, on the Transactions subtab.

This example uses dynamic mode, but you could also add the subrecord using standard mode. For general details about using either approach to add a sublist subrecord, see [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the record.

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

        // Set body fields on the record.

        rec.setValue({
            fieldId: 'entity',
            value: '1663'
        });

        rec.setValue({
            fieldId: 'location',
            value: '6'
        });

        rec.setValue({
            fieldId: 'memo',
            value: '456789'
        });

        // Create one line in the item sublist.

        rec.selectNewLine({
            sublistId: 'item',
            line: 0
        });

        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'item',
            value: '500'
        });

        rec.setCurrentSublistValue({
            sublistId: 'item',
            fieldId: 'quantity',
            value: '1'

```

```

});

// Create the subrecord for that line.

var subrec = rec.getCurrentSublistSubrecord({
    sublistId: 'item',
    fieldId: 'orderschedule'
});

// Set a field on the body of the subrecord.

subrec.setValue({
    fieldId: 'createpurchaseorder',
    value: 'LEAD'
});

// Create a line in the subrecord's sublist.

subrec.selectNewLine({
    sublistId: 'schedule',
});

subrec.setCurrentSublistValue({
    sublistId: 'schedule',
    fieldId: 'quantity',
    value: 1
});

var nextQuarter = new Date();
nextQuarter.setDate(nextQuarter.getDate() + 90);

subrec.setCurrentSublistValue({
    sublistId: 'schedule',
    fieldId: 'trandate',
    value: nextQuarter
});

// Save the line in the subrecord's sublist.

subrec.commitLine({
    sublistId: 'schedule'
});

// Save the line in the record's sublist

rec.commitLine({
    sublistId: 'item'
});

```

```

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });
    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

    return {
        afterSubmit: afterSubmit
    };
});

```

Example: Creating a Landed Cost Sublist Subrecord

The following example creates a vendor bill record. This example creates one line in the item sublist. It also sets the Landed Cost per Line option to true. With this configuration, it is possible to create a landed cost subrecord for each line.

To use this sample, the Landed Cost feature must be enabled at Setup > Company > Enable Features , on the Items & Inventory subtab.

This example uses standard mode, but you could also add the subrecord using dynamic mode. For general details about using either approach to add a sublist subrecord, see [Using SuiteScript 2.0 to Create a Subrecord in a Sublist Field](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the record.

        var rec = record.create({
            type : record.Type.VENDOR_BILL,
            isDynamic: false

```

```

});

// Set body fields on the record.

rec.setValue({
    fieldId: 'entity',
    value: '1663'
});

rec.setValue({
    fieldId: 'location',
    value: '6'
});

rec.setValue({
    fieldId: 'tranid',
    value: '101A'
});

// Set the Landed Cost per Line field to true.

rec.setValue({
    fieldId: 'landedcostperline',
    value: true
});

// Add an item to the Item sublist.

rec.insertLine({
    sublistId: 'item',
    line: 0
});

// Set values on the sublist line.

rec.setSublistValue({
    sublistId: 'item',
    fieldId: 'item',
    line: 0,
    value: '599'
});

rec.setSublistValue({
    sublistId: 'item',
    fieldId: 'quantity',
    line: 0,
    value: 1
});

rec.setSublistValue({
    sublistId: 'item',

```

```

        fieldId: 'location',
        line: 0,
        value: '6'
    });

    // Create the subrecord.

    var subrec = rec.getSublistSubrecord({
        sublistId: 'item',
        fieldId: 'landedcost',
        line: 0
    });

    // Add a line to the subrecord's Landed Cost Data sublist.

    subrec.insertLine({
        sublistId: 'landedcostdata',
        line: 0
    });

    // Set values on the subrecord's sublist line.

    subrec.setSublistValue({
        sublistId: 'landedcostdata',
        fieldId: 'costcategory',
        line: 0,
        value: 2
    });

    subrec.setSublistValue({
        sublistId: 'landedcostdata',
        fieldId: 'amount',
        line: 0,
        value: 17.85
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });
    } catch (e) {
        log.error({
            title: e.name,
            details: e.message
        });
    }

```



```

    }

    return {
      afterSubmit: afterSubmit
    };
  });
};

```

Using SuiteScript 2.0 to Edit a Subrecord that Occurs in a Sublist Field

In some cases, your script can make changes to a subrecord that occurs in a sublist field.

To edit a sublist subrecord, your script must use the [N/record Module](#). The script can use either dynamic or standard mode. For details, see the following sections:

- [Editing a Subrecord in Dynamic Mode](#)
- [Editing a Subrecord in Standard Mode](#)

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

Editing a Subrecord in Dynamic Mode

If your script uses dynamic mode, you can use the following procedure to edit a subrecord that occurs in a sublist field.

To edit a subrecord in dynamic mode:

1. Load the record.
2. Use the [Record.selectLine\(options\)](#) method to identify the sublist and line that contain the subrecord that you want to update.
3. Retrieve the subrecord with the [Record.getCurrentSublistSubrecord\(options\)](#) method. This method takes two arguments:
 - A `sublistId`.
 - A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.

For example, suppose you are working with an entity record, such as an employee or customer. You could use an expression like the following to load an address subrecord from the entity's Address sublist:

```

...
var addressSubrecord = rec.getCurrentSublistSubrecord({
  sublistId: 'addressbook',
  fieldId: 'addressbookaddress'
});
...

```

4. As appropriate, update body fields on the subrecord with the [Record.setValue\(options\)](#) method. For example, you could use an expression like the following to update a value on the address subrecord:

```

...
addressSubrecord.setValue({
  fieldId: 'city',
  value: 'St. Petersburg'
});

```

```
});
...
```

However, note that some subrecords do not have writable body fields.

5. If the subrecord has a sublist whose values you want to modify, use the following steps for each line you want to change:
 1. Identify the line you want to change with the [Record.selectLine\(options\)](#) method.
 2. For each value you want to change, use the [Record.setCurrentSublistValue\(options\)](#) method to identify the field and the new value.
 3. Save your changes to the subrecord's sublist line with the [Record.commitLine\(options\)](#) method.
6. Save the line that holds the subrecord with the [Record.commitLine\(options\)](#) method.
7. Save the record with the [Record.save\(options\)](#) method.

Note: For a full script sample that shows editing a sublist subrecord in dynamic mode, see [Example: Updating an Order Schedule Sublist Subrecord](#).

Editing a Subrecord in Standard Mode

If your script uses standard mode, you can use the following procedure to edit a subrecord that occurs in a sublist field.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To edit a subrecord in standard mode:

1. Load the record.
2. Retrieve the subrecord with the [Record.getSublistSubrecord\(options\)](#) method. This method takes three arguments:
 - A `sublistId`.
 - A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.
 - A line number, which identifies the sublist line that contains the subrecord you want to change.

For example, you could use an expression like the following to load an inventory detail subrecord from an item sublist:

```
...
inventoryDetailSubrecord = rec.getSublistSubrecord({
  sublistId: 'item',
  fieldId: 'inventorydetail',
  line: 0
});
...
```

3. Update body fields on the subrecord with the [Record.setValue\(options\)](#) method. Be aware that not all subrecords have writable body fields.
4. If the subrecord has a sublist whose values you want to modify, use the [Record.setSublistValue\(options\)](#) method to update the appropriate value. This method takes four arguments:
 - A `sublistId`.
 - A `fieldId`, which identifies the field you want to change.

- A line number, which identifies the sublist line you want to change.
- The new value.

For example, if you were updating an inventory detail subrecord, you could use the following expression to update the serial number on the first line of the inventory assignment sublist:

```
...
inventoryDetailSubrecord.setSublistValue({
  sublistId: 'inventoryassignment',
  fieldId: 'receiptinventorynumber',
  line: 0,
  value: '56789'
});
...
```

5. Save the record with the `save()` method.



Note: For a full script sample showing how to modify a subrecord in standard mode, see [Example: Updating an Address Subrecord](#).

Example: Updating an Order Schedule Sublist Subrecord

The following example loads an existing blanket purchase order record. It selects a line on the record's item sublist, retrieves the subrecord associated with that line, and makes changes to the subrecord.

To use this sample, you must meet the following prerequisites:

- The Blanket Purchase Order feature must be enabled at Setup > Company > Enable Features , on the Transactions subtab.
- The blanket purchase order record that you reference should have at least one line in the item sublist. The line should reference an existing order schedule subrecord.

This example uses dynamic mode, but you could also update the subrecord using standard mode. For general details about using either approach to update a sublist subrecord, see [Using SuiteScript 2.0 to Edit a Subrecord that Occurs in a Sublist Field](#). To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```
define([ 'N/record' ],

function(record) {

  function afterSubmit(context) {

    // Load the blanket purchase order record.

    var rec = record.load({
      type: record.Type.BLANKET_PURCHASE_ORDER,
      id: 3319,
      isDynamic: true
    });

    // Select the sublist and line.

    rec.selectLine({
```

```

        sublistId: 'item',
        line: 0
    });

    // Retrieve the subrecord.

    var subrec = rec.getCurrentSublistSubrecord({
        sublistId: 'item',
        fieldId: 'orderschedule'
    });

    // Select the appropriate line in the subrecord's sublist.

    subrec.selectLine({
        sublistId: 'schedule',
        line: 0
    });

    // Identify the field to be modified, and set new value.

    var nextQuarter = new Date();
    nextQuarter.setDate(nextQuarter.getDate() + 90);

    subrec.setCurrentSublistValue({
        sublistId: 'schedule',
        fieldId: 'trandate',
        value: nextQuarter
    });

    // Save the subrecord's sublist line.

    subrec.commitLine({
        sublistId: 'schedule'
    });

    // Save the item sublist line that contains the subrecord.

    rec.commitLine({
        sublistId: 'item'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record updated successfully',
            details: 'Id: ' + recId
        });
    }

```

```

    });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }
}

return {
    afterSubmit: afterSubmit
};
});

```

Example: Updating an Address Subrecord

The following example shows how to update an employee address. On the employee record, each address is stored in a subrecord that is associated with a line in the Address sublist.

This example uses standard mode, but you could also edit the subrecord using dynamic mode. For general details about using either approach to update a sublist subrecord, see [Using SuiteScript 2.0 to Edit a Subrecord that Occurs in a Sublist Field](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Load the record.

        var rec = record.load({
            type: record.Type.EMPLOYEE,
            id: 1863,
            isDynamic: false
        });

        // Retrieve the subrecord to be modified.

        var subrec = rec.getSublistSubrecord({
            sublistId: 'addressbook',
            fieldId: 'addressbookaddress',
            line: 0
        });
    }
}

```

```

    // Change a field on the subrecord.

    subrec.setValue({
        fieldId: 'addr1',
        value: '15 Main Street'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record updated successfully',
            details: 'Id: ' + recId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });

    }

    return {
        afterSubmit: afterSubmit
    };
});

```

Using SuiteScript 2.0 to Retrieve a Sublist Subrecord

In some cases, you may want to retrieve data from a subrecord that occurs in a sublist field.

To retrieve a sublist subrecord, use the [N/record Module](#). Your script can use either dynamic or standard mode. For details, see the following sections:

- [Retrieving a Sublist Subrecord in Dynamic Mode](#)
- [Retrieving a Subrecord in Standard Mode](#)

Retrieving a Sublist Subrecord in Dynamic Mode

If your script uses dynamic mode, you can use the following procedure to retrieve a subrecord that occurs in a sublist field.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To retrieve a sublist subrecord in dynamic mode:

1. Load the parent record.
2. Select the line that contains the subrecord with the [Record.selectLine\(options\)](#).
3. Retrieve the subrecord with the [Record.getCurrentSublistSubrecord\(options\)](#) method. This method takes two arguments:

- A `sublistId`.
- A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type `summary`.

For example, suppose you are working with an entity record, such as an employee or customer. You could use an expression like the following to load an address subrecord from the entity's Address sublist:

```
...
var addressSubrecord = rec.getCurrentSublistSubrecord({
  sublistId: 'addressbook',
  fieldId: 'addressbookaddress'
});
...
```

4. If you want to retrieve a value from the body of the subrecord, use the [Record.getValue\(options\)](#). For example, you could use an expression like the following to retrieve a detail from the body of an address subrecord:

```
...
var cityValue = addressSubrecord.getValue({
  fieldId: 'city'
});
...
```

5. If you want to retrieve a value from the subrecord's sublist, use the [Record.getSublistValue\(options\)](#) method.

Note: For a full script sample showing how to retrieve a sublist subrecord in dynamic mode, see [Example: Retrieving an Order Schedule Subrecord](#).

Retrieving a Subrecord in Standard Mode

If your script uses standard mode, use the following procedure to retrieve a sublist subrecord.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To retrieve a subrecord in standard mode

1. Load the parent record.
2. Retrieve the subrecord with the [Record.getSublistSubrecord\(options\)](#) method. This method takes three arguments:
 - A `sublistId`.
 - A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type `summary`.
 - A line number,

For example, you could use an expression like the following to load an order schedule subrecord from a blanket purchase order record:

```
...
var orderScheduleSubrecord = rec.getSublistSubrecord({
  sublistId: 'item',
  fieldId: 'orderschedule',
  line: 0
});
...
```

...

3. If you want to retrieve a value from the body of the subrecord, use the [Record.getValue\(options\)](#) method.
4. If you want to retrieve a value from the subrecord's sublist, use the [Record.getSublistValue\(options\)](#) method.

For example, the order schedule subrecord has a schedule sublist. If you wanted to retrieve the date value from the second line of the schedule sublist, you would use an expression like the following:

```
...
var dateValue = orderScheduleSubrecord.getSublistValue({
    sublistId: 'schedule',
    fieldId: 'trandate',
    line: 1
});
...
```



Note: For a full script sample showing how to retrieve a sublist subrecord in standard mode, see [Example: Retrieving an Address Subrecord](#).

Example: Retrieving an Order Schedule Subrecord

The following example loads a blanket purchase order record. It selects a line on the item sublist, then retrieves the order schedule associated with that line. It reads two values from the subrecord and prints them to the script deployment execution log.

If you want to use this script, you must meet the following prerequisites:

- The Blanket Purchase Order feature must be enabled at Setup > Company > Enable Features , on the Transactions subtab.
- The blanket purchase order record you reference must have at least one line in the Item sublist, and that line should include an order schedule subrecord.

This example uses dynamic mode, but you could also load the subrecord using standard mode. For general details about using either approach to retrieve a sublist subrecord, see [Using SuiteScript 2.0 to Retrieve a Sublist Subrecord](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Load the record.

        var rec = record.load({
            type: record.Type.BLANKET_PURCHASE_ORDER,
```



```

        id: 3120,
        isDynamic: true
    });

    // Retrieve the subrecord.

    rec.selectLine({
        sublistId: 'item',
        line: 0
    });

    var subrec = rec.getCurrentSublistSubrecord({
        sublistId: 'item',
        fieldId: 'orderschedule',
    });

    // Create a variable and initialize it to the
    // value of the trandate field.

    var dateValue = subrec.getSublistValue({
        sublistId: 'schedule',
        fieldId: 'trandate',
        line: 0
    });

    // Create a variable and initialize it to the
    // value of the memo field.

    var memoValue = subrec.getSublistValue({
        sublistId: 'schedule',
        fieldId: 'memo',
        line: 0
    });

    // Print the retrieved values to the execution log.

    try {

        log.debug({
            title: 'date value',
            details: 'date value: ' + dateValue
        });

        log.debug({
            title: 'memo value',
            details: 'memo value: ' + memoValue
        });

    } catch (e) {

        log.error({

```

```

        title: e.name,
        details: e.message
    });
}
}

return {
    afterSubmit: afterSubmit
};
});

```

Example: Retrieving an Address Subrecord

This example loads an employee record and selects a line on the address sublist. It retrieves a value from the sublist line that is not part of the subrecord. It also retrieves the address subrecord and reads one of the subrecord's fields. The script prints both values to the script deployment record's execution log.

This example uses standard mode, but you could also load the subrecord using dynamic mode. For general details about using either approach to retrieve a sublist subrecord, see [Using SuiteScript 2.0 to Retrieve a Sublist Subrecord](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

/**
 * @NApiVersion 2.x
 * @NScriptType usereventscript
 */

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Load the record.

        var rec = record.load({
            type: record.Type.EMPLOYEE,
            id: 1863,
            isDynamic: false
        });

        // Retrieve a value from the sublist line

        var labelValue = rec.getSublistValue({
            sublistId: 'addressbook',
            fieldId: 'label',
            line: 0
        });

        // Retrieve the subrecord associated with that same line.

```

```

var subrec = rec.getSublistSubrecord({
    sublistId: 'addressbook',
    fieldId: 'addressbookaddress',
    line: 0
});

// Create a variable to initialize it to the
// value of the subrecord's city field.

var cityValue = subrec.getValue({
    fieldId: 'city'
});

// Print the retrieved values to the execution log.

try {

    log.debug({
        title: 'label value',
        details: 'label value: ' + labelValue
    });

    log.debug({
        title: 'city value',
        details: 'city value: ' + cityValue
    });

} catch (e) {

    log.error({
        title: e.name,
        details: e.message
    });
}

return {
    afterSubmit: afterSubmit
};
});

```

Scripting Subrecords that Occur in Body Fields

In some cases, a subrecord is accessed through a body field on a record.

Most subrecords referenced from body fields behave in the same general way. However, there are some differences when working with subrecords that are used as the shipping or billing addresses on transactions.

For details, see the following sections:

- [Body Field Subrecords](#)

- [Transaction Shipping and Billing Addresses](#)

Body Field Subrecords

The following are examples of typical subrecords that are referenced from body fields:

- An assembly build record may need to include the serial numbers of each assembly being tracked. These details would be stored in an inventory detail subrecord referenced from a body field on the assembly build record.
- A subsidiary record may include a main address, a shipping address, and a return address. Each of these address subrecords is referenced by a field on the body of the record.
- The company information preferences page includes the preferred main address, shipping address, and return address for your organization. Each of these address subrecords is referenced by a field on the body of the record.

For details about working with these types of subrecords, see the following sections:

- [Using SuiteScript 2.0 to Create a Body Field Subrecord](#)
- [Using SuiteScript 2.0 to Update a Body Field Subrecord](#)
- [Using SuiteScript 2.0 to Retrieve a Body Field Subrecord](#)

Transaction Shipping and Billing Addresses

Some transactions can have body fields that represent shipping and billing addresses. Each of these fields contains an address subrecord. The process of interacting with these subrecords can be different from working with other body field subrecords. For details, see [Scripting Transaction Shipping and Billing Addresses](#).

Using SuiteScript 2.0 to Create a Body Field Subrecord

Some types of records have body fields that can reference subrecords. Scripting a subrecord that exists in a body field is slightly different from working with those that exist on sublists.

To create a subrecord, your script must use the [N/record Module](#). The script can use either dynamic or standard mode. For details on each approach, see the following sections:

- [Creating a Body Field Subrecord in Dynamic Mode](#)
- [Creating a Body Field Subrecord in Standard Mode](#)



Important: If you are scripting the shipping address or billing address on a transaction, see [Scripting Transaction Shipping and Billing Addresses](#).

Creating a Body Field Subrecord in Dynamic Mode

If your script uses dynamic mode, you can use the following procedure to create a subrecord in a body field.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To create a body field subrecord in dynamic mode:

1. If you want to add a new record, create it and set the required body fields. If you want to update an existing record, load the record.
2. Create the subrecord with the `Record.getSubrecord(options)`. This method takes one argument: A `fieldId`, which identifies the field on the record that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.

For example, you could use an expression like the following to create an inventory detail subrecord on an assembly build record:

```
...
var inventoryDetailSubrecord = rec.getSubrecord({
  fieldId: 'inventorydetail'
});
...
```

3. Set body fields on the subrecord with the `Record.setValue(options)`. Be aware that not all subrecords have writable body fields.
4. If the subrecord has a sublist, generally you are required to add at least one line to the sublist. For each line, use the following guidelines:
 - Create the line with the `Record.selectNewLine(options)`.
 - Set required values on the line with the `Record.setCurrentSublistValue(options)`.
 - Save the subrecord's sublist line with the `Record.commitLine(options)`.

For example, if you were creating an inventory detail subrecord, you could use the following expression to create a line on the subrecord's inventory assignment sublist:

```
...
inventoryDetailSubrecord.selectNewLine({
  sublistId: 'inventoryassignment',
});

inventoryDetailSubrecord.setCurrentSublistValue({
  sublistId: 'inventoryassignment',
  fieldId: 'receiptinventorynumber',
  value: '012345'
});

inventoryDetailSubrecord.commitLine({
  sublistId: 'inventoryassignment'
})
...
```

5. Save the record.



Note: For full script samples showing how to create a body field subrecord using dynamic mode, see [Example: Creating an Address on a Subsidiary Record](#) and [Example: Creating an Inventory Detail Subrecord on a Body Field](#).

Creating a Body Field Subrecord in Standard Mode

If your script uses standard mode, use the following procedure to create a subrecord on the body field of a record.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To create a body field subrecord in standard mode:

1. If you are adding a new record, create it and set the required body fields. If you are updating an existing record, load the record.
2. Create the subrecord with the [Record.getSubrecord\(options\)](#) method. This method takes one argument: A fieldId, which identifies the body field on the record that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type `summary`.

For example, you could use an expression like the following to create an address subrecord on a location record:

```
...
var addressSubrecord = rec.getSubrecord({
  fieldId: 'mainaddress'
});
...
```

3. Set body fields on the subrecord using the [Record.setValue\(options\)](#) method. Be aware that not all subrecords have writable body fields.
4. If the subrecord has a sublist, generally you are required to add at least one line to the sublist. For each line, use the following guidelines:
 - Create the line with the [Record.insertLine\(options\)](#) method.
 - Set required values on the line with the [Record.setSublistValue\(options\)](#) method.

For example, if you were creating an inventory detail subrecord, you could use the following expressions to create a line on the subrecord's inventory assignment sublist:

```
...
inventoryDetailSubrecord.insertLine({
  sublistId: 'inventoryassignment',
  line: 0
});

inventoryDetailSubrecord.setSublistValue({
  sublistId: 'inventoryassignment',
  fieldId: 'receiptinventorynumber',
  line: 0,
  value: '12345'
});
...
```

5. Save the record.

Example: Creating an Address on a Subsidiary Record

The following example shows how to create a subsidiary record that includes an address. In this case, the address data is contained in an address subrecord assigned to the `mainaddress` field.

To use this sample, you must meet the following prerequisites:

- You must have a OneWorld account.
- The value you use for the subsidiary record's name field must be unique in your system.

This example uses dynamic mode, but you could also add the subrecord using standard mode. For general details about using either approach, see [Using SuiteScript 2.0 to Create a Body Field Subrecord](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

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

        // Set body fields on the record.

        rec.setValue({
            fieldId: 'name',
            value: 'US Subsidiary'
        });

        rec.setValue({
            fieldId: 'state',
            value: 'CA'
        });

        // Create the address subrecord.

        var subrec = rec.getSubrecord({
            fieldId: 'mainaddress',
        });

        subrec.setValue({
            fieldId: 'city',
            value: 'San Mateo'
        });

        subrec.setValue({
            fieldId: 'state',
            value: 'CA'
        });

        subrec.setValue({
            fieldId: 'zip',
            value: '94403-2511'
        });
    }
});
```

```

        subrec.setValue({
            fieldId: 'addr1',
            value: '2955 Campus Drive'
        });

        subrec.setValue({
            fieldId: 'addr2',
            value: 'Suite 100'
        });

        // Save the record.

        try {
            var recId = rec.save();

            log.debug({
                title: 'Record created successfully',
                details: 'Id: ' + recId
            });
        } catch (e) {

            log.error({
                title: e.name,
                details: e.message
            });
        }

        return {
            afterSubmit : afterSubmit
        };
    });

```

Example: Creating an Inventory Detail Subrecord on a Body Field

The following example shows how to create an assembly build record that includes an inventory detail subrecord. In this case, the subrecord is contained in a body field called inventory detail.

To use this sample, you must meet the following prerequisites:

- The Advanced Bin / Numbered Inventory Management feature must be enabled at Setup > Company > Enable Features, on the Items & Inventory subtab.
- The item selected for the item body field should be a serialized assembly item.
- The receiptinventorynumber value must be unique in your system.

This example uses dynamic mode, but you could also add the subrecord using standard mode. For general details about using either approach to add a sublist subrecord, see [Using SuiteScript 2.0 to Create a Body Field Subrecord](#).

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

```

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

```



```

*/

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the record.

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

        // Set body fields.

        rec.setValue({
            fieldId: 'subsidiary',
            value: '1'
        });

        rec.setValue({
            fieldId: 'location',
            value: '6'
        });

        rec.setValue({
            fieldId: 'item',
            value: '699'
        });

        rec.setValue({
            fieldId: 'quantity',
            value: 1
        });

        // Create the subrecord.

        var subrec = rec.getSubrecord({
            fieldId: 'inventorydetail'
        });

        // Create a line on the subrecord's inventory assignment sublist.

        subrec.selectNewLine({
            sublistId: 'inventoryassignment',
        });

        subrec.setCurrentSublistValue({
            sublistId: 'inventoryassignment',
            fieldId: 'receiptinventorynumber',

```

```

        value: '012345'
    });

    subrec.commitLine({
        sublistId: 'inventoryassignment'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });
    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

    return {
        afterSubmit: afterSubmit
    };
});

```

Using SuiteScript 2.0 to Update a Body Field Subrecord

If the business logic of the parent record permits it, your script can load an existing body field subrecord and make changes to it.

To edit a subrecord, your script must use the [N/record Module](#). The script can use either dynamic or standard mode.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To update a body field subrecord:

1. Load the record.
2. Retrieve the subrecord with the [Record.getSubrecord\(options\)](#) method. This method takes one argument: A fieldId, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.

For example, you could use an expression like the following to load the inventory detail subrecord associated with an assembly build record:

```

...
var subrec = call.getSubrecord({

```

```

    fieldId: 'inventorydetail'
  });
  ...

```

3. Update body fields on the subrecord using the [Record.setValue\(options\)](#) method. Be aware that not all subrecords have writable body fields.
4. If the subrecord has a sublist whose values you want to change, you can:
 - If your script uses dynamic mode, use the following steps for each value you want to change:
 1. Identify the line you want to change with the [Record.selectLine\(options\)](#) method.
 2. For each value you want to change, use the [Record.setCurrentSublistValue\(options\)](#) method to identify the field and the new value.
 3. Save your changes to the subrecord's sublist line with the [Record.commitLine\(options\)](#) method.
 - If your script uses standard mode, use the [Record.setSublistValue\(options\)](#) method to update each field that you want to change. This field takes four arguments:
 - A sublistId.
 - A fieldId, which identifies the field on the sublist that contains the subrecord.
 - A line number, which identifies the sublist line that contains the subrecord you want to change.
 - The new value.
5. Save the record with the [Record.save\(options\)](#) method.



Note: For full script samples showing how to edit a body field subrecord using dynamic mode, see [Example: Editing a Body Field Address Subrecord](#) and [Example: Editing a Body Field Inventory Detail Subrecord](#).

Example: Editing a Body Field Address Subrecord

The following example loads a subsidiary record. It also loads and edits the address subrecord stored in the subsidiary's mainaddress body field.

To use this script, you must have a OneWorld account.

For general details on updating body field subrecords, see [Using SuiteScript 2.0 to Update a Body Field Subrecord](#).

```

/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 */
define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Load the record.

        var rec = record.load({

```

```

        type: record.Type.SUBSIDIARY,
        id: 1,
        isDynamic: true
    });

    // Load the subrecord.

    var subrec = rec.getSubrecord({
        fieldId: 'mainaddress'
    });

    // Make changes to one field.

    subrec.setValue({
        fieldId: 'addr1',
        value: '12331-A Riata Trace Parkway'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record updated successfully',
            details: 'Id: ' + recId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

    return {
        afterSubmit : afterSubmit
    };
});

```

Example: Editing a Body Field Inventory Detail Subrecord

The following example loads an assembly build record. It also loads the subrecord stored in the assembly build's Inventory Detail body field and saves a change to the subrecord.

To use this example, you must meet the following prerequisites:

- The Advanced Bin / Numbered Inventory Management feature must be enabled at Setup > Company > Enable Features, on the Items & Inventory subtab.
- The assembly build record that you load must already have an inventory detail subrecord.

- The new value you choose for receiptinventorynumber must be unique in your system.

For general details on updating body field subrecords, see [Using SuiteScript 2.0 to Update a Body Field Subrecord](#).

```
/**
 * @NApiVersion 2.x
 * @NScriptType UserEventScript
 */
define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Load the record.

        var rec = record.load({
            type: record.Type.ASSEMBLY_BUILD,
            id: 4918,
            isDynamic: true
        });

        // Load the subrecord.

        var subrec = rec.getSubrecord({
            fieldId: 'inventorydetail'
        });

        // Identify a line on the subrecord's sublist.

        subrec.selectLine({
            sublistId: 'inventoryassignment',
            line: 0
        });

        // Make changes to one sublist field.

        subrec.setCurrentSublistValue({
            sublistId: 'inventoryassignment',
            fieldId: 'receiptinventorynumber',
            value: '890123'
        });

        // Save the record.

        try {
            var recId = rec.save();

            log.debug({
                title: 'Record updated successfully',
```

```

        details: 'Id: ' + recId
    });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }
}

return {
    afterSubmit: afterSubmit
};
});

```

Using SuiteScript 2.0 to Retrieve a Body Field Subrecord

In some cases, you may want to retrieve data from a subrecord that occurs in a body field.

To retrieve a subrecord, your script can use either the [N/record Module](#) or the [N/currentRecord Module](#). Your script can use either dynamic or standard mode.

To learn about SuiteScript scripting modes, see [SuiteScript 2.0 – Standard and Dynamic Modes](#)

To use SuiteScript 2.0 to retrieve a body field subrecord:

1. Load the record.
2. Retrieve the subrecord with the `getSubrecord()` method. This method takes one argument: A `fieldId`, which identifies the field on the sublist that contains the subrecord. In the Records Browser, the field that holds the subrecord is always identified as a field of type summary.
For example, you could use an expression like the following to load an address subrecord stored in the body field of a location record:

```

...
var subrec = call.getSubrecord({
    fieldId: 'mainaddress'
});
...

```

3. If you want to retrieve a value from the body of the subrecord, use the `getValue()` method. For example, you could use an expression like the following to retrieve a detail from the body of an address subrecord:

```

...
var cityValue = subrec.getValue({
    fieldId: 'city'
});
...

```

4. If you want to retrieve a value from the subrecord's sublist, use the `getSublistValue()` method. For example, you could use an expression like the following to retrieve a detail from the sublist of an inventory detail subrecord:

```

...

```

```

var cityValue = subrec.getValue({
    sublistId: 'inventoryassignment',
    fieldId: 'receiptinventorynumber',
    line: 4,

});
...

```

Example: Retrieving a Body Field Address Subrecord

Many record types have body fields that reference address subrecords. A similar design exists with the Company Information page, which is accessed at Setup > Company > Company Information. Technically, this page is not a record, but you can interact with it as you would interact with a record. And like a record, the Company Information page includes body fields that reference subrecords. Specifically, it includes a mainaddress, shippingaddress, and returnaddress subrecord. The following example shows how to retrieve values from each of these addresses.

Note also that this page includes body fields labeled **state** and **country**. These fields are set independently of the subrecords. In SuiteScript, you interact with them as you would any standard body field.

To get the most from this example, you should first populate the Company Information page with values in the following fields:

- State
- County
- Address
- Shipping address
- Return address

For general details on retrieving body field subrecords, see [Using SuiteScript 2.0 to Update a Body Field Subrecord](#).

```

/**
 *@NApiVersion 2.x
 *@NScriptType UserEventScript
 */
define(['N/config', 'N/record'],

    function(config, record) {

        function afterSubmit(context) {

            // The Company Information page is not a standard record. You access it by
            // using the config.load method.

            var companyInfo = config.load({
                type: config.Type.COMPANY_INFORMATION
            });

            // Retrieve some of the preferences stored on the main part of the
            // Company Information page. You interact with these preferences as if
            // they were body fields.

```

```

var companyName = companyInfo.getValue({
    fieldId: 'companyname'
});

var employerId = companyInfo.getValue({
    fieldId: 'employerid'
});

// The Company Information page also includes some address fields on
// the body of the record. Retrieve these fields.

var state = companyInfo.getValue({
    fieldId: 'state'
});

var country = companyInfo.getValue({
    fieldId: 'country'
});

// Retrieve details from the subrecord that represents the main address.
// As a first step, instantiate the subrecord.

var mainAddress = companyInfo.getSubrecord ({
    fieldId: 'mainaddress'
});

// Retrieve values from the main address subrecord.

var mainAddressCity = mainAddress.getValue({
    fieldId: 'city'
});

var mainAddressState = mainAddress.getValue({
    fieldId: 'state'
});

// Retrieve details from the subrecord that represents the shipping address.
// To start, instantiate the subrecord.

var shippingAddress = companyInfo.getSubrecord('shippingaddress');

// Retrieve values from the subrecord.

var shippingAddressCity = shippingAddress.getValue({
    fieldId: 'city'
});

var shippingAddressState = shippingAddress.getValue({
    fieldId: 'state'
});

```



```

// Retrieve details from the subrecord that represents the return address.
// To start, instantiate the shipping address subrecord.

var returnAddress = companyInfo.getSubrecord('returnaddress');

// Retrieve values from the subrecord.

var returnAddressCity = returnAddress.getValue({
    fieldId: 'city'
});

var returnAddressState = returnAddress.getValue({
    fieldId: 'state'
});

// Write selected details to the log.

log.debug ({
    title: 'mainAddressState',
    details: mainAddressState
});

log.debug ({
    title: 'shippingAddressState',
    details: shippingAddressState
});

log.debug ({
    title: 'returnAddressState',
    details: returnAddressState
});

}

return {
    afterSubmit: afterSubmit
};

});

```

Scripting Transaction Shipping and Billing Addresses

A transaction's shipping address is stored in an address subrecord referenced from a body field on the transaction. A transaction's billing address is stored in the same way. The process of interacting with these subrecords is similar to how you work with other subrecords referenced by body fields. However, some differences exist. For details, see the following sections:

- [Addresses Can Be Sourced from Entity Records](#)
- [A Default Address May Be Used](#)

■ New Shipping and Billing Addresses Are Always Custom

 **Note:** See also [Understanding the Address Subrecord](#).

Addresses Can Be Sourced from Entity Records

Many transactions are associated with an entity. The entity record may already have addresses defined on its Address subtab. When working with a transaction, you can designate one of these existing addresses as the transaction's shipping or billing address.

If you choose to reference an existing address, your script does not have to create an address subrecord or use any subrecord methods. Instead, you can pick an address with a select field that is associated with the summary field. For example, to designate an existing address as the shipping address on a sales order, you use the `shipaddresslist` select field. This field identifies the subrecord being used by the transaction's `shippingaddress` summary field. For an example of how to select an existing address using the `shipaddresslist` select field, see [Example: Using SuiteScript 2.0 to Select an Existing Shipping Address](#).

A Default Address May Be Used

An entity can have a default shipping or billing address. For these types of entities, if you create a transaction and you don't set a value for shipping or billing address, the entity's default is used.

If the entity has a default shipping or billing address that you want to completely override, use `removeSubrecord()` to clear the summary field before you set the values for your new subrecord. As an alternative, you can set the `shipaddresslist` field to null before setting your new address values. For an example, see [Example: Using SuiteScript 2.0 to Create a New Shipping Address](#).

New Shipping and Billing Addresses Are Always Custom

In the UI, if you are creating a sales order and you want to enter a new shipping address, you must select either New or Custom in the Ship to Select field. If you select New, the address you enter is also saved to the customer record. If you select Custom, the new address is not saved to the customer record. It is saved only on the transaction.

By contrast, in SuiteScript, you cannot choose between New or Custom. Any new shipping or billing address created by your script is treated as a Custom address. The address is saved on the transaction, but it cannot be saved to the entity record.

Example: Using SuiteScript 2.0 to Create a New Shipping Address

The following script creates a sales order record with one line in the item sublist. It also creates a shipping address for the transaction. It overrides any default shipping address that might be defined on the customer record.

 **Note:** For details about working with a transaction's shipping or billing address, see [Scripting Transaction Shipping and Billing Addresses](#).

To use this sample, you must meet the following prerequisites:

- As already shown, when you create the new shipping address for the sales order, country field is set first (because the example uses dynamic mode – To learn about SuiteScript scripting modes,

see [SuiteScript 2.0 – Standard and Dynamic Modes](#)). For more information, see [Understanding the Address Subrecord](#).

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

        // Create the record.

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

        // Set body fields on the record.

        rec.setValue({
            fieldId: 'entity',
            value: '2163'
        });

        rec.setValue({
            fieldId: 'memo',
            value: '102A'
        });

        // To prevent the system from overriding the shipping address defined
        // in this script with a default shipping address (if one exists),
        // set the shipaddresslist field to null.

        rec.setValue({
            fieldId: 'shipaddresslist',
            value: null
        });

        // Create the subrecord.

        var subrec = rec.getSubrecord({
            fieldId: 'shippingaddress'
        });

        // Set values on the subrecord.
```

```

    // Set country field first when script uses dynamic mode

    subrec.setValue({
        fieldId: 'country',
        value: 'US'
    });

    subrec.setValue({
        fieldId: 'city',
        value: 'New York'
    });

    subrec.setValue({
        fieldId: 'state',
        value: 'New York'
    });

    subrec.setValue({
        fieldId: 'zip',
        value: '10018'
    });

    subrec.setValue({
        fieldId: 'addr1',
        value: '8 W 40th St.'
    });

    // Create a line in the item sublist.

    rec.selectNewLine({
        sublistId: 'item'
    });

    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'item',
        value: '100'
    });

    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'quantity',
        value: '11'
    });

    rec.commitLine({
        sublistId: 'item'
    });

    // Save the record.

    try {
        var recId = rec.save();
    }

```

```

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }
}

return {
    afterSubmit: afterSubmit
};
});

```

Example: Using SuiteScript 2.0 to Select an Existing Shipping Address

The following script creates a sales order record with one in line in the item sublist. It also sets the value of the shipping address field to the value of an address from the customer record. Because the script selects an existing address subrecord, it does not use subrecord methods. It identifies the address subrecord by internal ID.

Note: For details about working with a transaction's shipping or billing address, see [Scripting Transaction Shipping and Billing Addresses](#).

```

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

define([ 'N/record' ],

function(record) {

    function afterSubmit(context) {

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

        // Set body fields on the record.

        rec.setValue({
            fieldId: 'entity',
            value: '110'
        });
    }
}

```

```

    // Set the shipping address to the value of
    // an address on the customer record.

    rec.setValue({
        fieldId: 'shipaddresslist',
        value: '760'
    });

    // Create one line in the item sublist.

    rec.selectNewLine({
        sublistId: 'item'
    });

    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'item',
        value: '100'
    });

    rec.setCurrentSublistValue({
        sublistId: 'item',
        fieldId: 'quantity',
        value: '3'
    });

    rec.commitLine({
        sublistId: 'item'
    });

    // Save the record.

    try {
        var recId = rec.save();

        log.debug({
            title: 'Record created successfully',
            details: 'Id: ' + recId
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

    return {
        afterSubmit: afterSubmit
    };
});

```

Example: Using SuiteScript 2.0 to Retrieve a Shipping Address

The following user event script gets all fields from the shipping address of a new transaction record.

To use this script, deploy it on a sales order or another transaction that has a shippingaddress summary field. After you create a new transaction that includes a shipping address, the address should be written to the execution log of your script deployment record.

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

define(['N/record'],
  function(record) {
    function beforeSubmit(context) {
      if(context.type==context.UserEventType.CREATE) {

        var salesOrder = context.newRecord;

        var shipToAddress = salesOrder.getSubrecord({
          fieldId: 'shippingaddress'
        });

        log.debug({
          title: 'shipping address',
          details: JSON.stringify(shipToAddress)
        });
      }
    }

    return {
      beforeSubmit: beforeSubmit
    };
  });
```

Example: Using SuiteScript 2.0 to Retrieve one Value from a Shipping Address

The following script loads a sales order record. It retrieves a value from the shipping address and prints it to the execution log.

Note: For details about working with a transaction's shipping or billing address, see [Scripting Transaction Shipping and Billing Addresses](#).

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

define(['N/record'],
  function(record) {
```

```

function pageInit() {

    // Load the record.

    var rec = record.load({
        type: record.Type.SALES_ORDER,
        id: 5025,
        isDynamic: true
    });

    // Retrieve the subrecord.

    var subrec = rec.getSubrecord({
        fieldId: 'shippingaddress'
    });

    // Create a variable and initialize it to the
    // value of the subrecord's city field.

    var cityValue = subrec.getValue({
        fieldId: 'city'
    });

    // Print the value to the execution log.

    try {

        log.debug({
            title: 'city value',
            details: 'city value: ' + cityValue
        });

    } catch (e) {

        log.error({
            title: e.name,
            details: e.message
        });
    }

    return {
        pageInit: pageInit
    }
};

```


SuiteScript 2.0 Custom Pages

You can use SuiteScript 2.0 methods to create custom forms and list pages. See the following details:

- [SuiteScript 2.0 Custom Forms](#)
- [SuiteScript 2.0 Custom List Pages](#)



Important: SuiteScript does not support direct access to the NetSuite UI through the Document Object Model (DOM). You should only access the NetSuite UI by using SuiteScript APIs.

SuiteScript 2.0 Custom Forms

For information about using SuiteScript 2.0 to create custom forms, see:

- [Supported Script Types for Custom Form Creation](#)
- [Supported UI Components for Custom Forms](#)
- [Positioning Fields on Forms](#)
- [Sample Custom Form Script](#)

Supported Script Types for Custom Form Creation

You can use the following script types to create custom forms:

- **Suitelet:** Suitelets provide the most flexibility for creating a form. Suitelets can process requests and responses directly, giving you control over the data included in a form. For information about Suitelets, see [SuiteScript 2.0 Suitelet Script Type](#).
- **Portlet:** Portlet scripts are rendered within dashboard portlets. For information about portlet scripts, see [SuiteScript 2.0 Portlet Script Type](#).
- **Before Load User Event Script:** User event scripts are executed on the server when a user performs certain actions on records. You can use a before load user event script to manipulate an existing form when a user edits a record. For information about before load user event scripts, see [SuiteScript 2.0 User Event Script Type](#) and [beforeLoad\(scriptContext\)](#).

Supported UI Components for Custom Forms

You can use the following elements in your custom forms.

Buttons

You can use a button to trigger specific actions, such as standard buttons for Reset or Submit. The Reset button removes edits from a form, and the Submit button saves edits to a record. You also can attach a client script to a form and trigger it with a custom button.

- For information about adding a custom button to a form, see [Form.addButton\(options\)](#).

- For information about adding a reset button to a form, see [Form.addResetButton\(options\)](#).
- For information about adding a submit button to a form, see [Form.addSubmitButton\(options\)](#).

Fields

Add a customized field to a form to collect specific information from a user. Use the [serverWidget.FieldType](#) enumeration to specify the field type. Most field types include basic error checking to ensure that a user correctly formats their data.

- For information about adding a custom field to a form, see [Form.addField\(options\)](#).

Configuring Fields

After you add a field to a form, you can customize the positioning of the fields on the form by placing them within a field group, or on a tab or subtab.

- For information about field positioning and layout, see [Positioning Fields on Forms](#).
- For information about placing a field on a tab or subtab, see [Steps for Adding a Tab to a Form](#).

You can define a variety of field configuration properties by setting enum values. The following are examples of commonly used field configuration properties.

- Use [serverWidget.FieldDisplayType](#) to indicate whether a field should be displayed and how it should appear. For example, you can disable fields, hide them, or make them read-only.
- Use [Field.isMandatory](#) to indicate whether a field is required.
- For other field configuration properties, see [serverWidget.Field](#).

Specialized Fields

The Secret Key field is a special field that can be used with the [N/crypto Module](#) to perform encryption or hashing.

- For information about using a Secret Key field, see [Form.addSecretKeyField\(options\)](#).

The Credential field is a special text field that you can use to store credentials, such as a password, used to invoke third-party services. Credential fields can be restricted to certain domains, scripts, and users. The data stored in a credential field is encrypted, not stored as clear text.

- For information about using a Credential field, see [Form.addCredentialField\(options\)](#).

Field Groups

Use field groups to organize and manage fields on a form. Some of the properties listed in [serverWidget.FieldGroup](#) can provide additional field group customization.

- For details, see [Form.addFieldGroup\(options\)](#).

Tabs and Subtabs

You can add tabs or subtabs to a form to organize large groups of fields so that a user can navigate the form more easily.

- For information about adding a tab to a form, see [Steps for Adding a Tab to a Form](#) and [Form.addTab\(options\)](#).

Sublists

A sublist is a list of child records that can be added to a parent record form.

- For information about adding a sublist to a form, see [Steps for Adding a Sublist to a Form](#), and [Form.addSublist\(options\)](#).

Page Links

A page link on a form can be either a breadcrumb link or a crosslink. Breadcrumb links display a series of links leading to the location of the current page. Crosslinks are used for navigation, including links such as Forward, Back, List, Search, and Customize.

- For more information about page link types, see [serverWidget.FormPageLinkType](#).
- For more information about adding page links to a form, see [Form.addPageLink\(options\)](#).

Positioning Fields on Forms

You can add field groups to position fields together with other closely related fields on custom forms. Use the properties in [serverWidget.FieldGroup](#) to add field groups to forms. You also can use [Field.updateLayoutType\(options\)](#) to define the positioning and placement of fields.

Note: Some field group properties are not supported for field groups on forms, such as properties for collapsing and hiding field groups.

The following screenshot shows a single column field group above a double column field group.

The screenshot displays a form titled "Customer Information" with "Submit" and "Reset" buttons. It is divided into two main sections, each highlighted with a red border:

- User Information (Single Column):** Contains fields for TITLE (dropdown menu with "Mr." selected), FIRST NAME (required), LAST NAME (required), and EMAIL.
- Company Information (Double Column):** Contains fields for COMPANY (with "Company Name" as a placeholder), PHONE NUMBER, and WEBSITE.

Field Layout Type

The [serverWidget.FieldLayoutType](#) enumerations contain values used by [Field.updateLayoutType\(options\)](#) to position fields outside of a field group and on the same row.

You can use the OUTSIDE value to position a field outside of a field group. You can further refine the field's position by using the OUTSIDEABOVE and OUTSIDEBELOW value to position a field above or below a field group. The following screenshot displays three text fields positioned outside of a field group by setting these Field Layout Type values.

The following code sample illustrates how to set Field Layout values to produce the field placement in the screenshot.

```
//Add additional code
...
var outsideAbove = form.addField({
  id: 'outsideabovefield',
  type: serverWidget.FieldType.TEXT,
  label: 'Outside Above Field'
});
outsideAbove.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
});
...
//Add additional code
```

You can use the STARTROW, MIDROW, and ENDROW enumerations to position fields together in the same row. For example, you can group similar fields together, such as first name and last name. The following screenshot displays three text fields positioned together using these Field Layout Type values.

The following code sample illustrates how to set Field Layout values to produce the field placement in the screenshot.

```
//Add additional code
...
var startRow = form.addField({
  id: 'startrow',
  type: serverWidget.FieldType.TEXT,
  label: 'STARTROW',
  container: 'usergroup'
});

startRow.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.STARTROW
});

var midRow = form.addField({
  id: 'midrow',
  type: serverWidget.FieldType.TEXT,
  label: 'MIDROW',
  container: 'usergroup'
});

midRow.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.MIDROW
});

var endRow = form.addField({
  id: 'endrow',
  type: serverWidget.FieldType.TEXT,
  label: 'ENDROW',
  container: 'usergroup'
});

endRow.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.ENDROW
});

...
//Add additional code
```

Field Break Type

The `serverWidget.FieldBreakType` enumeration contains the values set by `Field.updateBreakType(options)` to define how fields are divided across columns and rows on forms.

- Use the `STARTCOL` value to move a field to a new column. The following screenshot shows the difference between the EMAIL field when the break type is set to `NONE` (top image) and when it is set to `STARTCOL` (bottom image).

The following code sample illustrates how to set the `FieldBreakType` values to produce the field placement in the screenshot.

```
//Add additional code
...
var email = form.addField({
  id: 'emailfield',
  type: serverWidget.FieldType.EMAIL,
  label: 'Email',
  container: 'usergroup'
});

email.updateBreakType({
  breakType : serverWidget.FieldBreakType.STARTCOL
});
...
//Add additional code
```

Use the `STARTROW` value to move a field to a new row. The `STARTROW` value can only be used on fields that are positioned outside of a field group. The `NONE` value is the default configuration, and does not move a field to a new row. For information about how to position a field outside of a field group, see [serverWidget.FieldLayoutType](#) and [Field.updateLayoutType\(options\)](#).

The following screenshot shows the difference between a field with a break type set to `STARTROW` and `NONE`.

The following code sample illustrates how to set the `FieldBreakType` to `STARTROW` to produce the field placement in the screenshot.

```
//Add additional code
... outsideAbove = form.addField({
  id: 'outsideabovefield',
  type: serverWidget.FieldType.TEXT,
  label: 'OUTSIDEABOVE'
});

outsideAbove.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.OUTSIDEABOVE
});

outsideAbove.updateBreakType({
  layoutType: serverWidget.FieldLayoutType.STARTROW
});
...
//Add additional code
```

Sample Custom Form Script

The following screenshot displays a form created by a Suitelet. You can use the sample code provided below to create this form.

The screenshot shows a web form titled "Customer Information" with a "More" link in the top right corner. The form is organized into three main sections, each with a header bar and a "Submit" button at the bottom.

- User Information:** Contains a "TITLE" dropdown menu (set to "Mr."), and text input fields for "FIRST NAME", "LAST NAME", and "EMAIL".
- Company Information:** Contains text input fields for "COMPANY", "PHONE NUMBER", "WEBSITE", and "WEBSITE2".
- Payment Information:** This section is highlighted with a dark blue header. It includes a "Payment Card" dropdown menu (set to "Payment Card 1"), "EXPIRY DATE" and "EXPIRY YEAR" dropdowns (set to "Jan" and "2020"), and a "CREDIT CARD NUMBER" text input field. A "Transaction Record" link is also visible.

At the bottom of the form, there are "Submit" and "Reset" buttons.

Steps for Creating a Custom Form with SuiteScript 2.0

Before proceeding, you must create a script file. To create this file, do one of the following:

- If you want to copy and paste the completed script directly from this help topic, go to the [Complete Custom Form Script Sample](#).
- If you want to walk through the steps for creating the script, complete the following procedure.

To create a custom form, complete the following steps:

1. Create a Suitelet and add the required JSDoc tags.

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

2. Add the define function to load the [N/ui/serverWidget Module](#) module.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
  function(serverWidget) {
```

3. To enable a Submit button, create an onRequest function. In the onRequest function create an if statement for the context request method and set it to 'GET'. After the if statement, create a return statement to support the GET request method.

Note: This sample is divided into four sections, as noted in the code comments. Each section identifies where to include different parts of the sample code. Section two of the sample code is referenced in [Steps for Adding a Tab to a Form](#). Section three of the sample code is referenced in [Steps for Adding a Sublist to a Form](#) sections.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
  function(serverWidget) {
    function onRequest(context){
      if(context.request.method === 'GET'){
        context.response.writePage(form);
        // Section One - Forms - See "Steps for Creating a Form"
        // Section Two - Tabs - See "Steps for Adding a Tab to a Form"
        // Section Three - Sublist - See "Steps for Adding a Sublist to a Form"
      }else{
        // Section Four - Output - Used in all sections
      }
    }
    return {
    }
  }
);
```

4. In section one, call the createForm method using [serverWidget.createForm\(options\)](#).

```
var form = serverWidget.createForm({
```



```

    title: 'Enter Customer Information'
  });
};

```

5. In section one, create Field Groups to organize the fields using [Form.addFieldGroup\(options\)](#). Use the properties defined in [serverWidget.FieldGroup](#) to control the appearance and behavior of the field groups.

```

var usergroup = form.addFieldGroup({
  id : 'usergroup',
  label : 'User Information'
});
usergroup.isSingleColumn = true;

var companygroup = form.addFieldGroup({
  id : 'companygroup',
  label : 'Company Information'
});

```

6. In section one, use [Form.addResetButton\(options\)](#) and [Form.addSubmitButton\(options\)](#) to add the Reset and Submit buttons to a form. Change the text displayed on the buttons by modifying the `label` property.

```

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

form.addResetButton({
  label: 'Clear Fields'
});

```

7. In section one, use [Form.addField\(options\)](#) to create the fields you need. Make sure to define the `id`, `type`, and `label` properties correctly. Use the `container` property to specify the field group.

Use the properties defined in [serverWidget.Field](#) to change the behavior and properties of the fields. This sample uses [Field.isMandatory](#) to define several required fields, and [Field.defaultValue](#) to set the default value of the Company name field.

Some fields, such as the email and phone fields, provide basic error checking to make sure the field contents meet the requirements for the field. For more information, see [serverWidget.FieldType](#).

Note: If you are modifying an existing form or adding custom fields, make sure you add the `custpage_` prefix to the identifier to prevent conflicts with existing standard fields.

```

var select = form.addField({
  id: 'titlefield',
  type: serverWidget.FieldType.SELECT,
  label: 'Title',
  container: 'usergroup'
});

select.addSelectOption({
  value: 'Mr.',
  text: 'Mr.'
});

```

```

select.addSelectOption({
    value: 'MS.',
    text: 'Ms.'
});

select.addSelectOption({
    value: 'Dr.',
    text: 'Dr.'
});

var fname = form.addField({
    id: 'fnamefield',
    type: serverWidget.FieldType.TEXT,
    label: 'First Name',
    container: 'usergroup'
});
fname.isMandatory = true;

var lname = form.addField({
    id: 'lnamefield',
    type: serverWidget.FieldType.TEXT,
    label: 'Last Name',
    container: 'usergroup'
});
lname.isMandatory = true;

form.addField({
    id: 'emailfield',
    type: serverWidget.FieldType.EMAIL,
    label: 'Email',
    container: 'usergroup'
});

var companyname = form.addField({
    id: 'companyfield',
    type: serverWidget.FieldType.TEXT,
    label: 'Company',
    container: 'companygroup'
});
companyname.defaultValue = 'Company Name';

form.addField({
    id: 'phonefield',
    type: serverWidget.FieldType.PHONE,
    label: 'Phone Number',
    container: 'companygroup'
});

form.addField({
    id: 'urlfield',
    type: serverWidget.FieldType.URL,
    label: 'Website',
    container: 'companygroup'
});

```

8. In section four, define a variable for each field in this sample.

```
var delimiter = /\u0001/;
var titleField = context.request.parameters.titlefield;
var fnameField = context.request.parameters.fnamefield;
var lnameField = context.request.parameters.lnamefield;
var emailField = context.request.parameters.emailfield;
var companyField = context.request.parameters.companyfield;
var phoneField = context.request.parameters.phonefield;
var urlField = context.request.parameters.urlfield;
```

9. In section four, create an output statement using `context.response.write`. This statement demonstrates the Submit button by generating an output displaying the variables created from the field data.

```
context.response.write('You have entered:'
  + '<br/> Name: ' + titleField + ' ' + fnameField + ' ' + lnameField
  + '<br/> Email: ' + emailField
  + '<br/> Company: ' + companyField
  + '<br/> Phone: ' + phoneField + ' Website: ' + urlField);
```

Steps for Adding a Tab to a Form

This sample uses the code created in [Sample Custom Form Script](#) to show how to add tab and subtab UI components to a form.

When you add a tab to a form, consider the following limitations:

- You cannot add a single tab. If you create a single tab, the contents of the tab appear on the form without the tab bar.
 - A tab does not appear until you have assigned an object to the form. After you have added a field, you can set the `container` property to indicate where the field appears on the tab.
1. In section two, use `Form.addTab(options)` to add two tabs to the form. You can use `Tab.helpText` to add inline help text to guide users.

Note: A value for `tab.id` must be all lowercase and contain no spaces. If you are adding a tab to an existing page, include the prefix `custpage` to prevent conflicts with existing standard tabs.

```
// Section Two - Tabs

var tab1 = form.addTab({
  id : 'tab1id',
  label : 'Payment'
});
tab1.helpText = 'Help Text Goes Here';

var tab2 = form.addTab({
  id : 'tab2id',
  label : 'Inventory'
});
```

2. In section two, use `Form.addSubtab(options)` to add two subtabs. Set the `tab` property to `tab1id` to position the subtab on the first tab you created.

```
form.addSubtab({
  id : 'subtab1id',
  label : 'Payment Information',
  tab: 'tab1id'
});

form.addSubtab({
  id : 'subtab2id',
  label : 'Transaction Record',
  tab: 'tab1id'
});
```

3. In section two, add more fields to the form. To assign a field to a tab, use the `container` property to indicate the tab where you want to place the field.

The following sample adds a Credential field to the form. Credential fields are text fields used to store credentials in NetSuite. Notice that when you click the Submit button in this sample, the credit card number is encrypted and is not displayed as clear text. For information about using Credential fields, see [Form.addCredentialField\(options\)](#).

```
//Subtab One Fields
var ccselect = form.addField({
  id: 'cctypefield',
  type: serverWidget.FieldType.SELECT,
  label: 'Credit Card',
  container: 'subtab1id'
});

ccselect.addSelectOption({
  value: 'PayCard0',
  text: 'Payment Card 0'
});

ccselect.addSelectOption({
  value: 'PayCard1',
  text: 'Payment Card 1'
});

ccselect.addSelectOption({
  value: 'PayCard2',
  text: 'Payment Card 2'
});

var expmonth = form.addField({
  id: 'expmonth',
  type: serverWidget.FieldType.SELECT,
  label: 'Expiry Month',
  container: 'subtab1id'
});

expmonth.updateLayoutType({
  layoutType: serverWidget.FieldLayoutType.STARTROW});
```

```
expmonth.addSelectOption({
    value: '01',
    text: 'Jan'
});

expmonth.addSelectOption({
    value: '02',
    text: 'Feb'
});

expmonth.addSelectOption({
    value: '03',
    text: 'Mar'
});

expmonth.addSelectOption({
    value: '04',
    text: 'Apr'
});

expmonth.addSelectOption({
    value: '05',
    text: 'May'
});

expmonth.addSelectOption({
    value: '06',
    text: 'Jun'
});

expmonth.addSelectOption({
    value: '07',
    text: 'Jul'
});

expmonth.addSelectOption({
    value: '08',
    text: 'Aug'
});

expmonth.addSelectOption({
    value: '09',
    text: 'Sep'
});

expmonth.addSelectOption({
    value: '10',
    text: 'Oct'
});

expmonth.addSelectOption({
    value: '11',
    text: 'Nov'
});
```

```

expmonth.addSelectOption({
    value: '12',
    text: 'Dec'
});

var expyear = form.addField({
    id: 'expyear',
    type: serverWidget.FieldType.SELECT,
    label: 'Expiry Year',
    container: 'subtablid'
});

expyear.updateLayoutType({
    layoutType: serverWidget.FieldLayoutType.ENDROW});

expyear.addSelectOption({
    value: '2020',
    text: '2020'
});

expyear.addSelectOption({
    value: '2019',
    text: '2019'
});

expyear.addSelectOption({
    value: '2018',
    text: '2018'
});

var credfield = form.addCredentialField({
    id : 'credfield',
    label : ' Credit Card Number',
    restrictToDomains : 'www.mysite.com',
    restrictToScriptIds : 'customscript_my_script',
    restrictToCurrentUser : false,
    container : 'subtablid'
});

credfield.maxLength = 32;

// Subtab Two fields
form.addField({
    id: 'transactionfield',
    type: serverWidget.FieldType.LABEL,
    label: 'Transaction History - Coming Soon',
    container: 'subtab2id'
});

//Tab Two fields
form.addField({
    id: 'inventoryfield',
    type: serverWidget.FieldType.LABEL,
    label: 'Inventory - Coming Soon',
    container: 'tab2id'
});

```

```
});
```

4. In section four, define a variable for each field you have added to the tabs. In this sample, these variables are used to demonstrate the function of the Submit button.

```
var ccField = context.request.parameters.cctypefield;
var ccNumber = context.request.parameters.credfield;
var expMonth = context.request.parameters.expmmonth;
var expYear = context.request.parameters.expyear;
```

5. In section four, update the output statement to display the variables created when the Submit button is clicked.

```
context.response.write('You have entered:');
+ '<br/> Name: ' + titleField + ' ' + fnameField + ' ' + lnameField
+ '<br/> Email: ' + emailField
+ '<br/> Company: ' + companyField
+ '<br/> Phone: ' + phoneField + ' Website: ' + urlField
+ '<br/> Credit Card: ' + ccField
+ '<br/> Number: ' + ccNumber
+ '<br/> Expiry Date: ' + expMonth + '/' + expYear);
```

Steps for Adding a Sublist to a Form

This sample uses the code created in [Steps for Adding a Tab to a Form](#) to show how to add a sublist to a form.

1. In section three, use `Form.addSublist(options)` to add a sublist to the form. Set the `tab` property to `tab2id` to position the sublist on the second tab. This sublist is an inline editor type sublist. For information about sublist types, see [serverWidget.SublistType](#).

```
var sublist = form.addSublist({
  id : 'sublistid',
  type : serverWidget.SublistType.INLINEEDITOR,
  label : 'Inventory',
  tab: 'tab2id'
});
```

2. In section three, use `Sublist.addButton(options)` to add a button to the sublist. In this sample the button is not functional, but you can use the `functionName` property to specify the function triggered when the button is clicked.

```
sublist.addButton({
  id : 'buttonId',
  label : 'Print ',
  functionName: '' // Add the function triggered on button click
});
```

3. In section three, use `Sublist.addField(options)` to add fields to the sublist. For more information about supported field types, see [serverWidget.FieldType](#).

Note: `Sublist.addField(options)` does not support the inline HTML field type.

```
sublist.addField({
```

```

        id : 'datefieldid',
        type : serverWidget.FieldType.DATE,
        label : 'Date'
    });

    sublist.addField({
        id : 'productfieldid',
        type : serverWidget.FieldType.TEXT,
        label : 'Product'
    });

    sublist.addField({
        id : 'qtyfieldid',
        type : serverWidget.FieldType.INTEGER,
        label : 'Quantity'
    });

    sublist.addField({
        id : 'upfieldid',
        type : serverWidget.FieldType.CURRENCY,
        label : 'Unit Cost'
    });

```

Complete Custom Form Script Sample

The following code creates a Suitelet that generates a custom form containing several field types, reset and submit buttons, tabs, and a sublist. For steps to create this script, see [Sample Custom Form Script](#).

```

/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
    function(serverWidget) {
        function onRequest(context) {
            if (context.request.method === 'GET') {

                // Section One - Forms - See "Steps for Creating a Form"
                var form = serverWidget.createForm({
                    title: 'Customer Information'
                });

                var usergroup = form.addFieldGroup({
                    id : 'usergroup',
                    label : 'User Information'
                });
                usergroup.isSingleColumn = true;

                var companygroup = form.addFieldGroup({
                    id : 'companygroup',
                    label : 'Company Information'
                });

                var select = form.addField({

```



```

        id: 'titlefield',
        type: serverWidget.FieldType.SELECT,
        label: 'Title',
        container: 'usergroup'
    });
    select.addSelectOption({
        value: 'Mr.',
        text: 'Mr.'
    });
    select.addSelectOption({
        value: 'MS.',
        text: 'Ms.'
    });
    select.addSelectOption({
        value: 'Dr.',
        text: 'Dr.'
    });
    });

    var fname = form.addField({
        id: 'fnamefield',
        type: serverWidget.FieldType.TEXT,
        label: 'First Name',
        container: 'usergroup'
    });
    fname.isMandatory = true;

    var lname = form.addField({
        id: 'lnamefield',
        type: serverWidget.FieldType.TEXT,
        label: 'Last Name',
        container: 'usergroup'
    });
    lname.isMandatory = true;

    form.addField({
        id: 'emailfield',
        type: serverWidget.FieldType.EMAIL,
        label: 'Email',
        container: 'usergroup'
    });

    var companyname = form.addField({
        id: 'companyfield',
        type: serverWidget.FieldType.TEXT,
        label: 'Company',
        container: 'companygroup'
    });
    companyname.defaultValue = 'Company Name';

    form.addField({
        id: 'phonefield',
        type: serverWidget.FieldType.PHONE,
        label: 'Phone Number',
        container: 'companygroup'
    });

```

```

});

form.addField({
  id: 'urlfield',
  type: serverWidget.FieldType.URL,
  label: 'Website',
  container: 'companygroup'
});

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

form.addResetButton({
  label: 'Reset'
});

// Section Two - Tabs - See "Steps for Adding a Tab to a Form"
var tab1 = form.addTab({
  id : 'tab1id',
  label : 'Payment'
});
tab1.helpText = 'Help Text Goes Here';

var tab2 = form.addTab({
  id : 'tab2id',
  label : 'Inventory'
});

form.addSubtab({
  id : 'subtab1id',
  label : 'Payment Information',
  tab: 'tab1id'
});

form.addSubtab({
  id : 'subtab2id',
  label : 'Transaction Record',
  tab: 'tab1id'
});

// Subtab One Fields
var ccselect = form.addField({
  id: 'cctypefield',
  type: serverWidget.FieldType.SELECT,
  label: 'Credit Card',
  container: 'subtab1id'
});

ccselect.addSelectOption({
  value: 'PayCard0',
  text: 'Payment Card 0'
});

ccselect.addSelectOption({

```

```

        value: 'PayCard1',
        text: 'Payment Card 1'
    });

    ccselect.addSelectOption({
        value: 'PayCard2',
        text: 'Payment Card 2'
    });

    var expmonth = form.addField({
        id: 'expmonth',
        type: serverWidget.FieldType.SELECT,
        label: 'Expiry Date:',
        container: 'subtablid'
    });
    expmonth.updateLayoutType({
        layoutType: serverWidget.FieldLayoutType.STARTROW});

    expmonth.addSelectOption({
        value: '01',
        text: 'Jan'
    });

    expmonth.addSelectOption({
        value: '02',
        text: 'Feb'
    });

    expmonth.addSelectOption({
        value: '03',
        text: 'Mar'
    });

    expmonth.addSelectOption({
        value: '04',
        text: 'Apr'
    });

    expmonth.addSelectOption({
        value: '05',
        text: 'May'
    });

    expmonth.addSelectOption({
        value: '06',
        text: 'Jun'
    });

    expmonth.addSelectOption({
        value: '07',
        text: 'Jul'
    });

    expmonth.addSelectOption({

```

```

        value: '08',
        text: 'Aug'
    });

    expmonth.addSelectOption({
        value: '09',
        text: 'Sep'
    });

    expmonth.addSelectOption({
        value: '10',
        text: 'Oct'
    });

    expmonth.addSelectOption({
        value: '11',
        text: 'Nov'
    });

    expmonth.addSelectOption({
        value: '12',
        text: 'Dec'
    });

    var expyear = form.addField({
        id: 'expyear',
        type: serverWidget.FieldType.SELECT,
        label: 'Expiry Year',
        container: 'subtablid'
    });

    expyear.updateLayoutType({
        layoutType: serverWidget.FieldLayoutType.ENDROW});

    expyear.addSelectOption({
        value: '2020',
        text: '2020'
    });

    expyear.addSelectOption({
        value: '2019',
        text: '2019'
    });

    expyear.addSelectOption({
        value: '2018',
        text: '2018'
    });

    var credfield = form.addCredentialField({
        id: 'credfield',
        label: 'Credit Card Number',
        restrictToDomains: 'www.mysite.com',
        restrictToScriptIds: 'customscript_my_script',
        restrictToCurrentUser: false
        container: 'subtablid'
    });

```

```

    };
    credfield.maxLength = 32;

    // Subtab two Fields
    form.addField({
      id: 'transactionfield',
      type: serverWidget.FieldType.LABEL,
      label: 'Transaction History - Coming Soon',
      container: 'subtab2id'
    });

    // Tab Two Fields
    form.addField({
      id: 'inventoryfield',
      type: serverWidget.FieldType.LABEL,
      label: 'Inventory - Coming Soon',
      container: 'tab2id'
    });

    // Section Three - Sublist - See "Steps for Adding a Sublist to a Form"

    var sublist = form.addSublist({
      id: 'sublistid',
      type: serverWidget.SublistType.INLINEEDITOR,
      label: 'Inline Sublist',
      tab: 'tab2id'
    });

    sublist.addButton({
      id: 'buttonId',
      label: 'Print ',
      functionName: '' // Add the function triggered on button click
    });

    // Sublist Fields
    sublist.addField({
      id: 'datefieldid',
      type: serverWidget.FieldType.DATE,
      label: 'Date'
    });

    sublist.addField({
      id: 'productfieldid',
      type: serverWidget.FieldType.TEXT,
      label: 'Product'
    });

    sublist.addField({
      id: 'qtyfieldid',
      type: serverWidget.FieldType.INTEGER,
      label: 'Quantity'
    });

    sublist.addField({

```

```

        id : 'upfieldid',
        type : serverWidget.FieldType.CURRENCY,
        label : 'Unit Cost'
    });

    context.response.writePage(form);

} else {
    // Section Four - Output - Used in all sections
    var delimiter = /\u0001/;
    var titleField = context.request.parameters.titlefield;
    var fnameField = context.request.parameters.fnamefield;
    var lnameField = context.request.parameters.lnamefield;
    var emailField = context.request.parameters.emailfield;
    var companyField = context.request.parameters.companyfield;
    var phoneField = context.request.parameters.phonefield;
    var urlField = context.request.parameters.urlfield;

    var ccField = context.request.parameters.cctypefield;
    var ccNumber = context.request.parameters.credfield;
    var expMonth = context.request.parameters.expmnth;
    var expYear = context.request.parameters.expyear;

    context.response.write('You have entered:'
        + '<br/> Name: ' + titleField + ' ' + fnameField + ' ' + lnameField
        + '<br/> Email: ' + emailField
        + '<br/> Company: ' + companyField
        + '<br/> Phone: ' + phoneField + ' Website: ' + urlField
        + '<br/> Credit Card: ' + ccField
        + '<br/> Number: ' + ccNumber
        + '<br/> Expiry Date: ' + expMonth + '/' + expYear);
    }
}
return {
    onRequest: onRequest
};
});

```

SuiteScript 2.0 Custom List Pages

For information about using SuiteScript 2.0 to create custom list pages, see:

- [Supported Script Types for Custom List Page Creation](#)
- [Supported UI Components for Custom List Pages](#)
- [Sample Custom List Page Script](#)

Supported Script Types for Custom List Page Creation

You can use the following script types to create custom list pages:

- **Suitelet:** Suitelets provide the most flexibility for creating a list. Suitelets can process requests and responses directly, giving you control over the data included in a list. For information about Suitelets, see [SuiteScript 2.0 Suitelet Script Type](#).
- **Portlet:** Portlet scripts are rendered within dashboard portlets. For information about portlet scripts, see [SuiteScript 2.0 Portlet Script Type](#).

Supported UI Components for Custom List Pages

You can use any of the following components on your custom list page.

Buttons

Add a button to a list page to trigger custom functions.

- For information about adding a button, see [List.addButton\(options\)](#).

Columns

A column contains an editable or read-only field that displays a record value. Column properties are defined using the supported field types specified in [serverWidget.FieldType](#). An Edit column is a special column that adds links to edit and view each record or row in the list.

- For information about adding columns to a list page, see [List.addColumn\(options\)](#).
- For information about adding an edit column, see [List.addEditColumn\(options\)](#).

Note: Use a URL column to add a website to each row in a list. Use [ListColumn.setURL\(options\)](#) to specify the base URL for the website, and [ListColumn.addParamToURL\(options\)](#) to assign additional properties to a URL column.

Page Link

Page links on list pages can be either a breadcrumb link or a crosslink. Breadcrumb links display a series of links leading to the location of the current page. Crosslinks are used for navigation, including links such as Forward, Back, List, Search, and Customize.

- For information about page link types, see [serverWidget.FormPageLinkType](#).
- For information about adding page links, see [List.addPageLink\(options\)](#).

Rows

Each row in a list represents a single record. You can add a single row or multiple rows to a custom list page by specifying the column ID and value for each row.

- For more information on adding a single row, see [List.addRow\(options\)](#).
- For more information on adding multiple rows, see [List.addRows\(options\)](#).

Sample Custom List Page Script

The following screenshot displays a list page created by a Suitelet. You can use the sample code provided below to create this list yourself.

Purchase History			
Test			
Purchase History			
DATE	PRODUCT	QUANTITY	UNIT COST
02/12/2018	Widget	4	4.50
02/14/2018	Sprocket	6	11.50
02/16/2018	Gizmo	9	1.25

Steps for Creating a Custom List Page with SuiteScript 2.0

Before proceeding, you must create a script file. To create this file, you can do one of the following:

- If you want to copy and paste the completed script directly from this help topic, go to [Complete Custom List Page Sample Script](#).
- If you want walk through the steps for creating the script, complete the following procedure.

To create a list page, complete the following steps:

1. Create a Suitelet, and add the required JSDoc tags.

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

2. Add the define function to load the [N/ui/serverWidget Module](#) module.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
  function(serverWidget) {
```

3. To enable a Submit button, create an onRequest function. In the onRequest function create an if statement for the context request method, and set it to 'GET'. After the if statement, create a return statement to support the GET request method.

Note: This sample is divided into two sections, as noted in the code comments. Each section identifies where to include different parts of the sample code.

```
/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
  function(serverWidget) {
    function onRequest(context){
      if(context.request.method === 'GET'){
```



```

        // Section One - List - See 'Steps for Creating a List', Step Five
        // Section Two - Columns - See 'Steps for Creating a List', Step Seven
        context.response.writePage(list);
    }else{
    }
}
return {
    onRequest: onRequest
}
});

```

4. In section one, create a list page using [serverWidget.createList\(options\)](#). You can use [serverWidget.ListStyle](#) to define the style of the list page.

```

var list = serverWidget.createList({
    title: 'Purchase History'
});
list.style = serverWidget.ListStyle.REPORT;

```

5. In section one, below the declaration of the list, use [List.addButton\(options\)](#) to create a button on the list. Use the `functionName` property to call the function triggered when the button is clicked.

```

list.addButton({
    id : 'buttonId',
    label : 'Test',
    functionName: '' // the function called when the button is pressed
});

```

6. In section two, use [List.addColumn\(options\)](#) to define the columns in the list. The column type is specified according to the [serverWidget.FieldType](#) enumeration.

Use the `align` property to justify the position of the column.



Important: The CHECKBOX field type is not supported by [List.addColumn\(options\)](#), and the MULTISELECT field is not supported by SuiteScript 2.0 Suitelets.

```

var datecol = list.addColumn({
    id : 'column1',
    type : serverWidget.FieldType.DATE,
    label : 'Date',
    align : serverWidget.LayoutJustification.RIGHT
});

list.addColumn({
    id : 'column2',
    type : serverWidget.FieldType.TEXT,
    label : 'Product',
    align : serverWidget.LayoutJustification.RIGHT
});

list.addColumn({
    id : 'column3',
    type : serverWidget.FieldType.INTEGER,
    label : 'Quantity',

```

```

        align : serverWidget.LayoutJustification.RIGHT
    });

    list.addColumn({
        id : 'column4',
        type : serverWidget.FieldType.CURRENCY,
        label : 'Unit Cost',
        align : serverWidget.LayoutJustification.RIGHT
    });

```

7. In section two, use `List.addColumn(options)` to define a single row on the list. Or use `List.addColumn(options)` to define multiple rows at once.

```

list.addRow({
    row : { column1 : '02/12/2018', column2 : 'Widget', column3: '4', column4:'4.50' }
});

```

```

list.addRows({
    rows :[{column1 : '02/12/2018', column2 : 'Widget', column3: '4', column4:'4.50'},
        {column1 : '02/14/2018', column2 : 'Sprocket', column3: '6', column4:'11.50'},
        {column1 : '02/16/2018', column2 : 'Gizmo', column3: '9', column4:'1.25'}]
});

```

Complete Custom List Page Sample Script

The following code creates a Suitelet that generates a custom list page. For steps to create this script, see [Sample Custom List Page Script](#).

```

/**
 * @NApiVersion 2.x
 * @NScriptType Suitelet
 */
define(['N/ui/serverWidget'],
    function(serverWidget) {
        function onRequest(context){
            if(context.request.method === 'GET'){
                // Section One - List - See 'Steps for Creating a List', Step Five
                var list = serverWidget.createList({
                    title: 'Purchase History'
                });

                list.style = serverWidget.ListStyle.REPORT;

                list.addButton({
                    id : 'buttonId',
                    label : 'Test',
                    functionName: '' // the function called when the button is pressed
                });

                // Section Two - Columns - See 'Steps for Creating a List', Step Seven
                var datecol = list.addColumn({
                    id : 'column1',

```

```

        type : serverWidget.FieldType.DATE,
        label : 'Date',
        align : serverWidget.LayoutJustification.RIGHT
    });

    list.addColumn({
        id : 'column2',
        type : serverWidget.FieldType.TEXT,
        label : 'Product',
        align : serverWidget.LayoutJustification.RIGHT
    });

    list.addColumn({
        id : 'column3',
        type : serverWidget.FieldType.INTEGER,
        label : 'Quantity',
        align : serverWidget.LayoutJustification.RIGHT
    });

    list.addColumn({
        id : 'column4',
        type : serverWidget.FieldType.CURRENCY,
        label : 'Unit Cost',
        align : serverWidget.LayoutJustification.RIGHT
    });

    list.addRows({
        rows : [{column1 : '05/30/2018', column2 : 'Widget', column3: '4', column4:'4.50'},
                {column1 : '05/30/2018', column2 : 'Sprocket', column3: '6', column4:'11.50'},
                {column1 : '05/30/2018', column2 : 'Gizmo', column3: '9', column4:'1.25'}]
    });
    context.response.writePage(list);
}
}
return {
    onRequest: onRequest
}
});

```