

SuiteScript 1.0 API Reference



Copyright © 2005, 2019, 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.

Send Us Your Feedback

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

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

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

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

To report software issues, contact NetSuite Customer Support.

Table of Contents

SuiteScript 1.0 API Overview	1
SuiteScript Functions	2
Record APIs	11
Subrecord APIs	38
Field APIs	43
Sublist APIs	55
Search APIs	79
Scheduling APIs	88
Execution Context APIs	96
UI Builder APIs	100
Application Navigation APIs	105
Date APIs	116
DateTime Time Zone APIs	118
Currency APIs	122
Encryption APIs	124
XML APIs	125
File APIs	132
Error Handling APIs	135
Communication APIs	136
Configuration APIs	145
SuiteFlow APIs	147
Portlet APIs	150
SuiteAnalytics APIs	152
User Credentials APIs	154
Job Manager APIs	155
SuiteScript Objects	158
Standard Objects	158
nlobjConfiguration	159
nlobjContext	164
nlobjCredentialBuilder(string, domainString)	180
nlobjCSVImport	184
nlobjDuplicateJobRequest	187
nlobjEmailMerger	190
nlobjError	193
nlobjFile	196
nlobjFuture	205
nlobjJobManager	206
nlobjLogin	207
nlobjMergeResult	209
nlobjPivotColumn	209
nlobjPivotRow	212
nlobjPivotTable	214
nlobjPivotTableHandle	215
nlobjRecord	216
nlobjReportColumn	250
nlobjReportColumnHierarchy	251
nlobjReportDefinition	251
nlobjReportForm	257
nlobjReportRowHierarchy	257
nlobjRequest	258
nlobjResponse	263
nlobjSearch	272
nlobjSearchColumn(name, join, summary)	289

nlobjSearchFilter	297
nlobjSearchResult	301
nlobjSearchResultSet	304
nlobjSelectOption	306
nlobjSubrecord	307
UI Objects	309
nlobjAssistant	309
nlobjAssistantStep	328
nlobjButton	332
nlobjColumn	334
nlobjField	336
nlobjFieldGroup	346
nlobjForm	349
nlobjList	369
nlobjPortlet	374
nlobjSubList	382
nlobjTab	389
nlobjTemplateRenderer	390
SuiteScript 1.0 API - Alphabetized Index	394

SuiteScript 1.0 API Overview

The SuiteScript API documentation is organized in a few different ways. Depending on how you wish to access the information, see any of the following links:

- [SuiteScript Functions](#) – Organizes the entire SuiteScript API into functional categories. Links to all functions **and** objects are provided.
- [SuiteScript Objects](#) – Defines all objects in the SuiteScript API
- [SuiteScript 1.0 API - Alphabetized Index](#) – Provides an alphabetized list of all SuiteScript functions and objects. If you prefer viewing APIs as an alphabetized list, click this link.



Important: If you are using SuiteScript 1.0 for your scripts, consider converting these scripts to SuiteScript 2.0. Use SuiteScript 2.0 to take advantage of new features, APIs, and functionality enhancements. For more information, see the help topic [SuiteScript 2.0 Advantages](#).

Important Things to Note:

- The SuiteScript API lets you programmatically extend NetSuite beyond the capabilities offered through SuiteBuilder point-and-click customization. However, a sound understanding of general NetSuite customization principles will help you when writing SuiteScript. If you have no experience customizing NetSuite using SuiteBuilder, it is worth seeing [SuiteBuilder Overview](#).
- Most SuiteScript APIs pass record, field, sublist, tab, search filter, and search column IDs as arguments. In the NetSuite Help Center, see the help topic [Working with the SuiteScript Records Browser](#) to learn how to access all supported internal IDs.
- In your SuiteScript code, all record, field, sublist, tab, search join, search field, and search column IDs must be in **lowercase**.
- If you are new to SuiteScript and have no idea how to get a script to run in your NetSuite account, you should start here: [SuiteScript 1.0 - The Basics](#).



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.

SuiteScript Functions

SuiteScript Functions Overview



Important: If you are not familiar with the SuiteScript API, we recommend you see [SuiteScript 1.0 API Overview](#).

This documentation organizes all SuiteScript functions into the functional categories listed below. Note that some APIs appear in more than one category. For example, `nlapiLookupField()` appears in both the [Field APIs](#) and [Search APIs](#) categories, however it is documented only one time.

- Working with entire record object – see [Record APIs](#)
- Working with subrecords – see [Subrecord APIs](#)
- Working with fields on a record – see [Field APIs](#)
- Working with sublists on a record – see [Sublist APIs](#)
- Searching in NetSuite – see [Search APIs](#)
- Submitting scheduled scripts for processing – see [Scheduling APIs](#)
- Getting context information about a script, a user, an account – see [Execution Context APIs](#)
- Building a NetSuite-looking user interface in a Suitelet – see [UI Builder APIs](#)
- Setting application navigation – see [Application Navigation APIs](#)
- Working with Date and String objects – see [Date APIs](#)
- Working with alternate time zones — see [Time Zone APIs](#)
- Working with currency – see [Currency APIs](#)
- Adding security to your application – see [Encryption APIs](#)
- Working with XML – see [XML APIs](#)
- Working with new or existing files – see [File APIs](#)
- Adding error handling – see [Error Handling APIs](#)
- Communicating to external systems from within NetSuite — see [Communication APIs](#)
- Configuring your NetSuite account - see [Configuration APIs](#)
- Interacting with the NetSuite Workflow (SuiteFlow) Manager - see [SuiteFlow APIs](#)
- Working with dashboard portlets - see [Portlet APIs](#)
- Working with NetSuite Analytics - see [SuiteAnalytics APIs](#)
- Changing Currently Logged-in User Credentials - see [User Credentials APIs](#)
- Working with the NetSuite Job Manager - see [Job Manager APIs](#)

Record APIs

Functions
nlapiAttachRecord(type, id, type2, id2, attributes)
nlapiCalculateTax()
nlapiCopyRecord(type, id, initializeValues)
nlapiCreateCSVImport()
nlapiCreateEmailMerger(templateId)

nlapiCreateRecord(type, initializeValues)
nlapiDeleteRecord(type, id, initializeValues)
nlapiDetachRecord(type, id, type2, id2, attributes)
nlapiGetNewRecord()
nlapiGetOldRecord()
nlapiGetRecordId()
nlapiGetRecordType()
nlapiLoadRecord(type, id, initializeValues)
nlapiMergeRecord(id, baseType, baseId, altType, altId, fields)
nlapiMergeTemplate(id, baseType, baseId, altType, altId, fields)
nlapiPrintRecord(type, id, mode, properties)
nlapiSubmitCSVImport(nlobjCSVImport)
nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)
nlapiTransformRecord(type, id, transformType, transformValues)
nlapiVoidTransaction(transactionType, recordId)
Objects
nlobjCSVImport object and all methods
nlobjRecord object and all methods

Subrecord APIs


Functions
nlapiCreateCurrentLineItemSubrecord(sublist, fldname)
nlapiCreateSubrecord(fldname)
nlapiEditCurrentLineItemSubrecord(sublist, fldname)
nlapiEditSubrecord(fldname)
nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)
nlapiRemoveSubrecord(fldname)
nlapiViewCurrentLineItemSubrecord(sublist, fldname)
nlapiViewLineItemSubrecord(sublist, fldname, linenum)
nlapiViewSubrecord(fldname)
Objects
nlobjRecord object and its “subrecord-related” methods
nlobjSubrecord object and all methods

Field APIs

Functions
nlapiDisableField(fldnam, val)
nlapiGetField(fldnam)
nlapiGetFieldText(fldnam)
nlapiGetFieldTexts(fldnam)
nlapiGetFieldValue(fldnam)
nlapiGetFieldValues(fldnam)
nlapiInsertSelectOption(fldnam, value, text, selected)
nlapiLookupField(type, id, fields, text)
nlapiRemoveSelectOption(fldnam, value)
nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)
nlapiSetFieldTexts (fldname, txts, firefieldchanged, synchronous)
nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)
nlapiSetFieldValues (fldnam, value, firefieldchanged, synchronous)
nlapiSubmitField(type, id, fields, values, doSourcing)
Objects
nlobjField object and all methods
nlobjRecord and all methods
nlobjSelectOption and all methods

Sublist APIs

Functions
nlapiCancelLineItem(type)
nlapiCommitLineItem(type)
nlapiDisableLineItemField(type, fldnam, val)
nlapiFindLineItemMatrixValue(type, fldnam, val, column)
nlapiFindLineItemValue(type, fldnam, val)
nlapiGetCurrentLineItemIndex(type)
nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)
nlapiGetCurrentLineItemText(type, fldnam)
nlapiGetCurrentLineItemValue(type, fldnam)

<code>nlapiGetCurrentLineItemValues(type, fldnam)</code>
<code>nlapiGetLineItemCount(type)</code>
<code>nlapiGetLineItemField(type, fldnam, linenum)</code>
<code>nlapiGetLineItemMatrixField(type, fldnam, linenum, column)</code>
<code>nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)</code>
<code>nlapiGetLineItemText(type, fldnam, linenum)</code>
<code>nlapiGetLineItemValue(type, fldnam, linenum)</code>
<code>nlapiGetLineItemValues(type, fldname, linenum)</code>
<code>nlapiGetMatrixCount(type, fldnam)</code>
<code>nlapiGetMatrixField(type, fldnam, column)</code>
<code>nlapiGetMatrixValue(type, fldnam, column)</code>
<code>nlapiInsertLineItem(type, line)</code>
<code>nlapiInsertLineItemOption(type, fldnam, value, text, selected)</code>
<code>nlapiIsLineItemChanged(type)</code>
<code>nlapiRefreshLineItems(type)</code>
<code>nlapiRemoveLineItem(type, line)</code>
<code>nlapiRemoveLineItemOption(type, fldnam, value)</code>
<code>nlapiSelectLineItem(type, linenum)</code>
<code>nlapiSelectNewLineItem(type)</code>
<code>nlapiSetCurrentLineItemMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)</code>
<code>nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)</code>
<code>nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)</code>
<code>nlapiSetCurrentLineItemValues(type, fldnam, values, firefieldchanged, synchronous)</code>
<code>nlapiSetLineItemValue(type, fldnam, linenum, value)</code>
<code>nlapiSetMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)</code>
Objects
<code>nlobjSubList</code> and all methods
 Note: Also see Subrecord APIs for a list of functions that can be used to create and access a subrecord from a sublist field.

Search APIs

Functions

nlapiCreateSearch(type, filters, columns)
nlapiLoadSearch(type, id)
nlapiLookupField(type, id, fields, text)
nlapiSearchDuplicate(type, fields, id)
nlapiSearchGlobal(keywords)
nlapiSearchRecord(type, id, filters, columns)
Objects
nlobjSearch and all methods
nlobjSearchColumn object and all methods
nlobjSearchFilter object and all methods
nlobjSearchResult and all methods
nlobjSearchResultSet and all methods

Scheduling APIs

Functions
nlapiScheduleScript(scriptId, deployId, params)
nlapiSetRecoveryPoint()
nlapiYieldScript()

Execution Context APIs

Functions
nlapiGetContext()
nlapiGetDepartment()
nlapiGetLocation()
nlapiGetRole()
nlapiGetSubsidiary()
nlapiGetUser()
nlapiLogExecution(type, title, details)
Objects
nlobjContext object and all methods

UI Builder APIs

Functions

nlapiCreateAssistant(title, hideHeader)
nlapiCreateForm(title, hideNavbar)
nlapiCreateList(title, hideNavbar)
nlapiCreateTemplateRenderer()
Objects
nlobjAssistant object and all methods
nlobjAssistantStep object and all methods
nlobjButton object and all methods
nlobjColumn object and all methods
nlobjField object and all methods
nlobjFieldGroup and all methods
nlobjForm object and all methods
nlobjList object and all methods
nlobjPortlet object and all methods
nlobjSubList object and all methods
nlobjTab object and all methods
nlobjTemplateRenderer object and all methods

Application Navigation APIs

Functions
nlapiRequestURL(url, postdata, headers, callback, httpMethod)
nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)
nlapiResolveURL(type, identifier, id, displayMode)
nlapiSetRedirectURL(type, identifier, id, editmode, parameters)
Objects
nlobjRequest object and all methods
nlobjResponse object and all methods

Date APIs

Functions
nlapiAddDays(d, days)
nlapiAddMonths(d, months)

<code>nlapiDateToString(d, format)</code>
<code>nlapiStringToDate(str, format)</code>

Time Zone APIs

Functions
<code>nlapiGetCurrentLineItemDateTimeValue(type, fieldId, timeZone)</code>
<code>nlapiGetDateTimeValue(fieldId, timeZone)</code>
<code>nlapiGetLineItemDateTimeValue(type, fieldId, lineNum, timeZone)</code>
<code>nlapiSetCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)</code>
<code>nlapiSetDateTimeValue(fieldId, dateTime, timeZone)</code>
<code>nlapiSetLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)</code>
Objects
<code>nlobjRecord</code> object and its “datetime time zone conversion” methods

Currency APIs

Functions
<code>nlapiExchangeRate(sourceCurrency, targetCurrency, effectiveDate)</code>
<code>nlapiFormatCurrency(str)</code>

Encryption APIs

Functions
<code>nlapiEncrypt(s, algorithm, key)</code>

XML APIs

Functions
<code>nlapiEscapeXML(text)</code>
<code>nlapiSelectNode(node, xpath)</code>
<code>nlapiSelectNodes(node, xpath)</code>
<code>nlapiSelectValue(node, xpath)</code>
<code>nlapiSelectValues(node, path)</code>
<code>nlapiStringToXML(text)</code>
<code>nlapiValidateXML(xmlDocument, schemaDocument, schemaFolderId)</code>

<code>nlapiXMLToString(xml)</code>
<code>nlapiXMLToPDF(xmlstring)</code>

File APIs

Functions
<code>nlapiCreateFile(name, type, contents)</code>
<code>nlapiDeleteFile(id)</code>
<code>nlapiLoadFile(id)</code>
<code>nlapiSubmitFile(file)</code>
Objects
<code>nlobjFile</code> object and all methods

Error Handling APIs

Functions
<code>nlapiCreateError(code, details, suppressNotification)</code>
Objects
<code>nlobjError</code> object and all methods

Communication APIs

Functions
<code>nlapiOutboundSSO(id)</code>
<code>nlapiRequestURL(url, postdata, headers, callback, httpMethod)</code>
<code>nlapiSendCampaignEmail(campaigneventid, recipientid)</code>
<code>nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo)</code>
<code>nlapiSendFax(author, recipient, subject, body, records, attachments)</code>

Configuration APIs

Functions
<code>nlapiLoadConfiguration(type)</code>
<code>nlapiSubmitConfiguration(name)</code>
Objects
<code>nlobjConfiguration</code> object and all methods

SuiteFlow APIs

Functions
nlapiInitiateWorkflow(recordtype, id, workflowid, initialvalues)
nlapiInitiateWorkflowAsync(recordType, id, workflowId, initialValues)
nlapiTriggerWorkflow(recordtype, id, workflowid, actionid, stateid)

Portlet APIs

Functions
nlapiRefreshPortlet()
nlapiResizePortlet()

SuiteAnalytics APIs

Functions
nlapiCreateReportDefinition()
nlapiCreateReportForm(title)
Objects
nlobjPivotColumn object and all methods
nlobjPivotRow object and all methods
nlobjPivotTable object and all methods
nlobjPivotTableHandle object and all methods
nlobjReportColumn object and all methods
nlobjReportColumnHierarchy object and all methods
nlobjReportDefinition object and all methods
nlobjReportForm object and all methods
nlobjReportRowHierarchy object and all methods

User Credentials APIs

Function
nlapiGetLogin()
Objects
nlobjLogin object and all methods

Job Manager APIs

Function
nlapiGetJobManager(jobType)
Objects
nlobjJobManager object and all methods
nlobjDuplicateJobRequest and all methods
nlobjFuture and all methods

Record APIs

For an overview of NetSuite records, see the help topic [SuiteScript 1.0 Working with Records](#). For information about sourcing, as it pertains to working with records, see [Understanding Sourcing in SuiteScript](#).

All APIs listed below are in alphabetical order.

- [nlapiAttachRecord\(type, id, type2, id2, attributes\)](#)
- [nlapiCopyRecord\(type, id, initializeValues\)](#)
- [nlapiCreateCSVImport\(\)](#)
- [nlapiCreateEmailMerger\(templateId\)](#)
- [nlapiCreateRecord\(type, initializeValues\)](#)
- [nlapiDeleteRecord\(type, id, initializeValues\)](#)
- [nlapiDetachRecord\(type, id, type2, id2, attributes\)](#)
- [nlapiGetNewRecord\(\)](#)
- [nlapiGetOldRecord\(\)](#)
- [nlapiGetRecordId\(\)](#)
- [nlapiGetRecordType\(\)](#)
- [nlapiLoadRecord\(type, id, initializeValues\)](#)
- [nlapiMergeRecord\(id, baseType, baseId, altType, altId, fields\)](#)
- [nlapiMergeTemplate\(id, baseType, baseId, altType, altId, fields\)](#)
- [nlapiPrintRecord\(type, id, mode, properties\)](#)
- [nlapiSubmitCSVImport\(nlobjCSVImport\)](#)
- [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#)
- [nlapiTransformRecord\(type, id, transformType, transformValues\)](#)
- [nlapiVoidTransaction\(transactionType, recordId\)](#)
- [nlobjCSVImport](#)
- [nlobjRecord](#)

nlapiAttachRecord(type, id, type2, id2, attributes)

Attaches a single record to another record. The following attachment relationships are supported:

- A Support Case to an Issue

- A Contact to any Customer, Partner, Vendor, Lead, Prospect, or Project
- A File to any transaction, item, activity, custom, or entity record
- A custom child record to a supported parent record
- An entity to a static entity group.

This API is supported in client, user event, scheduled, and Suitelet scripts.

For more information about the unit cost associated with this API, see the help topic [SuiteScript 1.0 API Governance](#).

Parameters

- **type** {string} [required] - The record ID for the type of record to attach. For a list of supported record types and their internal IDs, see the help topic [SuiteScript Supported Records](#).
To attach a file from the file cabinet to a record, set **type** to **file**.
- **id** {int} [required] - The internalId of the record to attach.
- **type2** {string} [required] - The record ID for the type of record that is receiving the attachment.
To attach an entity to a static entity group, set **type2** to **entitygroup**.
- **id2** {int} [required] - The internalId of the record that is receiving the attachment.
- **attributes** {hashtable} [optional] - Name/value pairs containing attributes for the attachment:
 - contact->company record: **role** (the contact role id used for attaching contact to customer/vendor/partner)
 - customrecord*->parent record: **field** (the custom field used to link child custom record to parent record)

Returns

- void

Since

- Version 2008.1

Example 1


The following example shows how to attach a Support Case record to an Issue record.

```
function testAttachment(request, response)
{
    //Define variables for nlapiAttachRecord
    var type = 'supportcase'; //Define record type for the record being attached
    var id = 2352; //Ensure id2 is a valid ID. An error is thrown if id2 is not valid.
    var type2 = 'issue'; //Define the record type for the record being attached to
    var id2 = 372; //Define the internal ID for this record
    var attributes = null;
    nlapiAttachRecord(type, id, type2, id2, attributes);
    response.write('done');
}
```

Example 2

The following sample shows how to attach and detach a child custom record from a parent record. Prior to running this script, a custom record (record id = customrecord15) was created. Next, a field was added


to the record (field id = custrecord_customer). The field was marked as a select/list field (source list = customer) and the **record is parent** was set.

 **Note:** This script assumes there is a record with id=1 and a customer with id=79.

```
var fld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
nlapiAttachRecord('customrecord15', 1, 'customer', 79, {'field' : 'custrecord_customer'})
var newFld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
nlapiDetachRecord('customrecord15', 1, 'customer', 79, {'field' : 'custrecord_customer'})
var finalFld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
```

Example 3

This sample shows how to attach a file object to a record (in this case a JPEG to a Customer record). After it is attached, the file appears on the Files tab of the Customer record.


 **Important:** Although the **file** record name is referenced in **nlapiAttachRecord**, you cannot yet reference **file** in other record-level APIs – such as **nlapiCreateRecord**, **nlapiSubmitRecord**, or **nlapiLoadRecord**. The File (**file**) record is not yet supported in this capacity.

```
var type = 'file'; // the record type for the record being attached
var id = 297; // the internal ID of an existing jpeg in the file cabinet
var type2 = 'customer'; // the record type for the record being attached to
var id2 = 406; // this is the internal ID for the customer
var attributes = null;
nlapiAttachRecord(type, id, type2, id2, attributes)
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiCalculateTax()

Supported in client script when the SuiteTax feature is enabled in the account. Calculates tax for the record currently open in the browser. Tax calculation results are stored within the record. Taxes are always calculated when the transaction is saved.

 **Note:** There is currently no governance cost for this function.

Parameters

- None

Returns

- void

Since

- 2015.2

Throws

- SSS_UNSUPPORTED_METHOD, if the SuiteTax feature is disabled, or the transaction type of the current record is non-taxable

- `SSS_TAX_REGISTRATION_REQUIRED`, if the subsidiary of the current record does not have a valid tax registration

Example

```
nlapiCalculateTax();
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiCopyRecord(type, id, initializeValues)

Initializes a new record using field data from an existing record. Note that this API simply creates a new instance of another record. After making changes to the copy, you must submit the copy (which is considered as a new record) to the database for your changes to be committed to NetSuite.

This API is supported in all script types. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.

Parameters

- **type** {string} [required] - The record internal ID name. For a list of supported record types and their internal IDs, see the help topic [SuiteScript Supported Records](#) in the NetSuite Help Center.
- **id** {int} [required] - The internalId for the record. If you are unsure how to find a record's internalId, see the help topic [Showing Record and Field IDs in Your Account](#).
- **initializeValues** {Object} [optional] - Contains an array of name/value pairs of defaults to be used during record initialization. For a list of record initialization types and the values they can take, see the help topic [SuiteScript 1.0 Record Initialization Defaults](#) in the NetSuite Help Center.
- **ignorecache** {boolean} [optional] - This option allows you to enable temporary, short-term storage of data. Default value is F for false, which means use caching. If T or true is used, this means the record is loaded exactly as it is in database.

Returns

- An [nlobjRecord](#) object of a copied record

Example

The following example initializes a new standalone copy of an invoice record from an existing one. Standalone copies are not linked to the original record.

```
var copyorder = nlapiCopyRecord('invoice', 659, {standalonecopy: 'T'});
partner.setFieldValue('currency', 2);
partner.setFieldValue('location', 6);
var copiedId = nlapiSubmitRecord(copyorder);
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateCSVImport()

Initializes a new record and returns an [nlobjCSVImport](#) object. You can then use the methods available on the returned record object to populate the object with the desired information. Next, you can pass

this object to [nlapiSubmitCSVImport\(nlobjCSVImport\)](#), which asynchronously imports the data from the returned object into NetSuite.

Note that this API cannot be used to 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.



Warning: This API is only supported for bundle installation scripts, scheduled scripts, and RESTlets. If you attempt to execute this API in another type of script, an error is returned.

Parameters

- None

Returns

- An [nlobjCSVImport](#) object to be passed as a parameter to [nlapiSubmitCSVImport\(nlobjCSVImport\)](#).

Since

- Version 2012.2

Examples

This sample uses a script ID to reference the import mapping and raw string for CSV data:

```
var mappingFileId = "CUSTIMPORTjob1"; //using script id of Saved CSV Import
var primaryFileAsString = "company name,isperson,subsidiary,externalid\ncompanytest001,FALSE,Parent Company,companytest001";

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);
job.setPrimaryFile(primaryFileAsString);
job.setOption("jobName", "job1Import");

//returns the internal id of the new job created in workqueue
var jobId = nlapiSubmitCSVImport(job);
```

This sample uses an internal ID to reference the import map and a CSV file internal ID to reference CSV data:

```
var mappingFileId = 2; //using internal id of Saved CSV Import
var primaryFile = nlapiLoadFile(73); //using the internal id of the file stored in the File Cabinet

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);
job.setPrimaryFile(primaryFile);
job.setOption("jobName", "job2Import");

//returns the internal id of the new job created in workqueue
var jobId = nlapiSubmitCSVImport(job);
```

This sample, which is a multi-file import, uses a script ID to reference the import map and CSV file internal IDs to reference CSV data, and provides a sublist identifier in the [setLinkedFile\(sublist, file\)](#) method:

```

var mappingFileId = "CUSTIMPORTentityMultiFile";

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);

//uploaded to File Cabinet <multifile_entity_primary.csv> with internal Id = 73
job.setPrimaryFile(nlapiLoadFile(73));

//uploaded to File Cabinet <multifile_entity_cust_address.csv> with internal Id = 74
job.setLinkedFile("addressbook", nlapiLoadFile(74));
job.setOption("jobName", "test_ImportMultiFileTransactionRecord-");

var jobId = nlapiSubmitCSVImport(job);

```

For more details about the methods used in these samples, see [nlobjCSVImport](#).

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateEmailMerger(templateId)

With scriptable e-mail templates (Freemarker templates), you can create dynamic templates for e-mail marketing campaigns and system e-mail. See the help topic [Scriptable Templates](#) for additional information.

This API is called as the first step in performing a mail merge with an existing scriptable e-mail template. The following record types are supported:

- Contact
- Case
- Customer
- Employee
- Partner
- Vendor
- All transaction types
- All custom records



Important: This function only supports scriptable email templates. It does not support CRMSDK templates.

System email templates supported by the now deprecated `nlapiMergeRecord` function are not supported by `nlapiCreateEmailMerger`. This function does not support system email templates. If you attempt to use this function with a system email template, you will receive an `SSS_MERGER_ERROR_OCCURRED` error with the message: "Incorrect template type".

To perform a mail merge with a scriptable email template:

1. Use `nlapiCreateEmailMerger(templateId)` to create an `nlobjEmailMerger` object. The function `nlapiCreateEmailMerger(templateId)` takes the record ID of a scriptable template as an argument. The object `nlobjEmailMerger` encapsulates the scriptable template.

```
var emailMerger = nlapiCreateEmailMerger(<templateId>);
```

2. Use the [nlobjEmailMerger](#) set methods to designate the records to perform the mail merge on.

```
emailMerger.setEntity(<entityType>, <entityId>);
emailMerger.setRecipient(<recipientType>, <recipientId>);
emailMerger.setSupportCase(<caseId>);
emailMerger.setTransaction(<transactionId>);
emailMerger.setCustomRecord(<recordType>, <recordId>);
```

3. Use the [nlobjEmailMerger.merge\(\)](#) method to perform the mail merge. The [merge\(\)](#) method returns an [nlobjMergeResult](#) object that contains the subject and body of the e-mail distribution.

Note: The [nlobjEmailMerger.merge\(\)](#) method has a governance of 20 usage units. The remaining APIs described here have no governance.

```
var mergeResult = emailMerger.merge();
```

4. Use the [nlobjMergeResult](#) methods to obtain the e-mail distribution's subject and body in string format.

```
var emailSubject = mergeResult.getSubject();
var emailBody = mergeResult.getBody();
```

Parameters

- `templateId {number} [required]` – Internal ID of the scriptable template you want to use.

Returns

- an [nlobjEmailMerger](#) object

Throws

- `SSS_MERGER_ERROR_OCCURRED` – Thrown if the email merger fails.

Note: Most input validation is performed after [nlobjEmailMerger.merge\(\)](#) is called. Therefore most errors are thrown at that point in the script.

Since

Version 2015 Release 1

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateRecord(type, initializeValues)

Initializes a new record and returns an [nlobjRecord](#) object containing all the default field data for that record type. You can then use the methods available on the returned record object to populate the record with the desired information.

The `nlapiCreateRecord` function must be followed by the [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#) function before the record is actually committed to the database.

This API is supported in all script types. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.

Note: Values for all required fields **must** be provided or a newly instantiated record cannot be submitted. See the help topic [SuiteScript Supported Records](#) in the NetSuite Help Center for records that support SuiteScript, their fields, and whether the fields are required for each record type. You can also refer to the NetSuite UI, which displays all required fields in yellow. There may be additional required fields when custom forms are used. Also, **Note** records cannot be created as standalone records. These records are always associated with another record. Similarly, **Message** records require an author and recipient to ensure that they are not created as standalone records.

Parameters

- **type** {string} [required] - The record internal ID name. For a list of supported record types and their internal IDs, see the help topic [SuiteScript Supported Records](#) in the NetSuite Help Center.
- **initializeValues** {Object} [optional] - Contains an array of name/value pairs of defaults to be used during record initialization. For a list of record initialization types and the values they can take, see the help topic [SuiteScript 1.0 Record Initialization Defaults](#) in the NetSuite Help Center.
- **ignorecache** {boolean} [optional] - This option allows you to enable temporary, short-term storage of data. Default value is F for false, which means use caching. If T or true is used, this means the record is loaded exactly as it is in database.

Returns

- An [nlobjRecord](#) object of a new record

Throws

- SSS_INVALID_RECORD_TYPE
- SSS_TYPE_ARG_REQD

Example 1

The following example initializes a new Opportunity record.

```
var record = nlapiCreateRecord('opportunity')
var defaultstatus = record.getFieldValue('entitystatus')
```

Example 2

In the next example, the **createTaskRecord** function causes a new task record to be created. This could be tied to an **afterSubmit** function of a user event and deployed to Opportunity records so that each time an Opportunity is created, a task is automatically created.

Note: You must use the [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#) function in conjunction with **nlapiCreateRecord** for the new record to be saved to the database.

```
function createTaskRecord()
{
    var taskTitle = 'Follow up regarding new Opportunity';
    var record = nlapiCreateRecord('task');
    record.setFieldValue('title', taskTitle);
    id = nlapiSubmitRecord(record, true);
}
```


Example 3

This example shows how to create a Message record, set its fields, and then submit the record.

```
var message = nlapiCreateRecord('message');
message.setFieldValue('entity', ...)
message.setFieldValue('message', ...)
//... set all the necessary fields
var internalId = nlapiSubmitRecord( message );
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiDeleteRecord(type, id, initializeValues)

Use this API to delete an existing record. This API is supported in all script types. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.



Warning: Use caution when using the **nlapiDeleteRecord** function in SuiteScript. Records deleted using **nlapiDeleteRecord** are **permanently** deleted from the NetSuite database.

Parameters

- **type** {string} [required] - The record internal ID name. For a list of supported record types and their internal IDs, see the help topic [SuiteScript Supported Records](#) in the NetSuite Help Center.
- **id** {int} [required] - The internalId for the record
- **initializeValues** {Object} [optional] - Contains an array of name/value pairs of defaults to be used during record initialization. For a list of record initialization types and the values they can take, see the help topic [SuiteScript 1.0 Record Initialization Defaults](#) in the NetSuite Help Center.
- **ignorecache** {boolean} [optional] - This option allows you to enable temporary, short-term storage of data. Default value is F for false, which means use caching. If T or true is used, this means the record is loaded exactly as it is in database.

Returns

- void

Throws

- SSS_INVALID_RECORD_TYPE
- SSS_TYPE_ARG_REQD
- SSS_INVALID_INTERNAL_ID
- SSS_ID_ARG_REQD

Example 1

The following example deletes a specific task record in the system.

```
var id = nlapiDeleteRecord('task', 5000);
```

Example 2

In the next example a resultant record set from a customer saved search is deleted. After the search is performed, methods on the [nlobjSearchResult](#) object take the desired action. In this example, the

`nlobjSearchResult.getRecordType` and `nlobjSearchResult.getId` methods are used to identify which records to delete.

```
function executeSavedSearch()
{
    var searchresults = nlapiSearchRecord('customer', 57, null, null);
    for (var i = 0; searchresults != null && i < searchresults.length; i++)
    {
        var searchresult = searchresults[i];
        nlapiDeleteRecord(searchresults[i].getRecordType(), searchresults[i].getId());
    }
}
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiDetachRecord(type, id, type2, id2, attributes)

Use this API to detach a single record from another record. The following detach relationships are supported:

- Issue detached from Support Case
- Contact detached from Customer | Partner | Vendor | Lead | Prospect | Project
- File detached from any transaction, item, activity, custom, or entity record
- Custom child record detached from supported parent record
- Entity detached from a static entity group. Note that if detaching an entity from a static entity group, you must specify **entitygroup** as the internal ID for the **type2** argument (see below).

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.

Parameters

- **type** {string} [required] - The record internal ID name for the record being detached. For a list of record names, see the column called "Record Internal ID" in [SuiteScript Supported Records](#).
- **id** {int} [required] - The record internalId for the record being detached
- **type2** {string} [required] - The record internal ID name for the record being detached from. Note that if detaching an entity from a static entity group, the internal ID for the entity group record type is **entitygroup**.
- **id2** {int} [required] - The record internalId for the record being detached from
- **attributes** {hashtable} [optional] - Name/value pairs containing attributes for the attachment:
 - `customrecord*->parent record: field` (the custom field used to link child custom record to parent record)

Returns

- void

Since

- Version 2008.1


Example 1

The following example shows how to detach an Issue record from a Support Case record.

```
function testDetach(request, response)
{
  //Define variables for nlapiDetachRecord
  var type = 'issue'; //Define the record type for the record being detached
  var id = 2352; //Define the internal ID for this record
  var type2 = 'supportcase'; //Define record type for the record being detached from
  var id2 = 372; //Ensure id2 is a valid ID. An error is thrown if id2 is not valid.
  var attributes = null;
  nlapiDetachRecord(type, id, type2, id2, attributes)
  response.write('done');
}
```

Example 2

The following sample shows how to attach and detach a child custom record from a parent record. Prior to running this script, a custom record (record id = customrecord15) was created. Next, a field was added to the record (field id = custrecord_customer). The field was marked as a select/list field (source list = customer) and the **record is parent** was set.

 **Note:** This script assumes there is a record with id=1 and a customer with id=79.

```
var fld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
nlapiAttachRecord('customrecord15', 1, 'customer', 79, {'field' : 'custrecord_customer'})
var newFld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
nlapiDetachRecord('customrecord15', 1, 'customer', 79, {'field' : 'custrecord_customer'})
var finalFld = nlapiLookupField('customrecord15', 1, 'custrecord_customer')
```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetNewRecord()

Available in **beforeLoad**, **beforeSubmit**, and **afterSubmit** user event scripts. You are not allowed to submit the current or previous record returned by **nlapiGetNewRecord**.

When triggered by an inline edit event (**type == xedit**), this function only returns the field and sublist line item values that were edited. For all other triggers, **nlapiGetNewRecord** returns all record object values.

Returns

- An **nlobjRecord** containing all the values being used for a write operation

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetOldRecord()

Available in **beforeLoad**, **beforeSubmit**, and **afterSubmit** user event scripts. You are not allowed to submit the current or previous record returned by **nlapiGetOldRecord**.

Returns

- An [nlobjRecord](#) containing all the values for the current record prior to the write operation
- Null if used on Workflow Action Scripts

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetRecordId()

Use this API to retrieve the **internalId** of the current record in a user event script. This API is available in client and user event scripts only.

Returns

- The integer value of the record whose form the user is currently on, or for the record that the current user event script is executing on. Note that the value of **-1** is returned if there is no current record or the current record is a new record.

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetRecordType()

Use this API to retrieve the record type **internal ID** of the current record in a user event script or a client script. If there is no current record type, the value of null will be returned.

Returns

- The record type **internal ID** as a string. Example return values are:
 - **salesorder** for a script executed on a Sales Order record
 - **customer** for a script executed on a Customer record
 - **promotioncode** for a script executed on the Promotion record

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiLoadRecord(type, id, initializeValues)

Loads an existing record from the system and returns an [nlobjRecord](#) object containing all the field data for that record. You can then extract the desired information from the loaded record using the methods available on the returned record object. This API is a core API. It is available in both client and server contexts.

This API is supported in all script types. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.



Important: Only records that support SuiteScript can be loaded using this API. In NetSuite Help Center, see the help topic [SuiteScript Supported Records](#) for a list of records that support SuiteScript. Also be aware that if a particular record instance has been locked by the [Lock Record Action](#) workflow action, you will be unable to load the record using the **nlapiLoadRecord** API.

Note that when using this API, you can:

- set the **type** parameter to 'inventoryitem' to load the following types of item records: inventoryitem, lotnumberedinventoryitem, serializedinventoryitem.

- set the **type** parameter to 'assemblyitem' to load the following types of item records: assemblyitem, lotnumberedasassemblyitem, serializedassemblyitem.
- set the **type** parameter to 'customer' to load the following types of entity records: customer, lead, prospect.
- set the **type** parameter to 'usereventscript', 'suitelet', 'scheduledscript', 'clientscript', 'restlet', 'massupdatescript', 'bundleinstallationscript', 'workflowactionscript', or 'portlet' to load script records.
- set the **type** parameter to 'scriptdeployment' to load script deployment records.

Parameters

- **type** {string} [required] - The record internal ID name. This parameter is case-insensitive. In the NetSuite Help Center, see the help topic [SuiteScript Supported Records](#). Use the values listed in the column "Record Internal ID".
- **id** {int} [required] - internalId for the record, for example 555 or 78.
- **initializeValues** {Object} [optional] - Contains an array of name/value pairs of defaults to be used during record initialization. For a list of record initialization types and the values they can take, see the help topic [SuiteScript 1.0 Record Initialization Defaults](#) in the NetSuite Help Center.
- **ignorecache** {boolean} [optional] - This option allows you to enable temporary, short-term storage of data. Default value is F for false, which means use caching. If T or true is used, this means the record is loaded exactly as it is in database.

Returns

- An [nlobjRecord](#) object of an existing NetSuite record. This function returns the record object **exactly** as the record appears in the system. Therefore, in **beforeLoad** user event scripts, if you attempt to change a field and load the record simultaneously, the change will not take effect.

Throws

- SSS_INVALID_RECORD_TYPE
- SSS_TYPE_ARG_REQD
- SSS_INVALID_INTERNAL_ID
- SSS_ID_ARG_REQD

Example 1

The following example loads a customer record from the system. After the record is loaded, the script uses the `nlobjRecord.getFieldValue` method to return the value of the **phone** field. Finally, the number of line items on the Address sublist are returned using the `nlobjRecord.getLineItemCount` method.

```
var record = nlapiLoadRecord('customer', 100)
var phone = record.getFieldValue('phone')
var numberOfAddresses = record.getLineItemCount('addressbook');
```

Example 2

In the next example, the search described in the section on `nlapiSearchRecord(type, id, filters, columns)` is performed, but each search result object is loaded using the `nlapiLoadRecord(type, id, initializeValues)` function. Then the `getRecordType()` and `getId()` `nlobjRecord` methods are used to retrieve specific information about each record.

```
function executeSearch()
```

```

{
  var rec = '';
  var searchresults = nlapiSearchRecord( 'customer', null, null, null );
  for ( var i = 0; i < Math.min( 500, searchresults.length ); i++)
  {
    var record = nlapiLoadRecord(searchresults[i].getRecordType(),
      searchresults[i].getId() );
    rec = rec + record.getRecordType() ;
    rec = rec + ' -Record ID = ' + record.getId();
  }
  nlapiSendEmail(312, 312, 'customerRecordLoaded', rec, null);
}

```

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiPrintRecord(type, id, mode, properties)

Returns an [nlobjFile](#) object containing the PDF or HTML document. This API is supported in user event, scheduled, and Suitelet scripts.



Note: There is a 10MB limitation to the size of the file that can be accessed using this API.

Direct manipulation of the print URL is **not** supported.

There are two primary use cases for **nlapiPrintRecord**:

1. Send email or fax attachments using either [nlapiSendEmail\(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo\)](#) or [nlapiSendFax\(author, recipient, subject, body, records, attachments\)](#). See [Example 1](#) and [Example 2](#).

For example, you can create a PDF or HTML object of a transaction or statement and then send the object as an attachment. This would be useful when sending out monthly collections notices for customers with overdue invoices.

2. Stream PDF/HTML documents to the server (for example, to maintain an archive of statement/transactions on your server). [Example 3](#).



Important: **nlapiPrintRecord** is **not** supported in client scripting. This is a server-side-only API. Also note that this function does not send transactions or statements to a printer to be printed. It also does not launch Adobe Acrobat if the **mode** specified is PDF.

If the Advanced PDF/HTML Templates feature is enabled in your account, this function supports the use of advanced templates. If you associate an advanced template with the custom form saved for a transaction and use this API to print the transaction, the advanced template is used to format the printed transaction. For details about this feature, see the help topic [Advanced PDF/HTML Templates](#).

Parameters

- **type** {string} [required] - Print operation type. Can be any of the following:
 - TRANSACTION
 - STATEMENT
 - PACKINGSLIP
 - PICKINGTICKET

- BILLOFMATERIAL
- **id** {int} [required] - The internal ID of the transaction or statement being printed
- **mode** {string} [optional] - The output type: PDF|HTML|DEFAULT. DEFAULT uses the user/company preference for print output
- **properties** {hashtable} [optional] - Name/value pairs used to configure the print operation.
 - TRANSACTION: formnumber
 - STATEMENT: openonly (T|F), startdate, statementdate, formnumber, subsidiary
 - PACKINGSLIP: formnumber, itemfulfillment
 - PICKINGTICKET: formnumber, shipgroup, location
 - **incustlocale** {boolean} [optional] - This setting applies when advanced templates are used. If not set, this argument defaults to false, which means that the document is printed in the default company language. If set to true, the document is printed 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.

Returns

- **nlobjFile** object

Since

- Version 2008.1

Example 1

In the following sample a PDF object is created from a specific transaction. This object is then sent as an attachment using **nlapiSendEmail**.

```
function printTrans()
{
  //print the transaction to a PDF file object
  var file = nlapiPrintRecord('TRANSACTION', 1799, 'DEFAULT', null);

  //send the PDF as an attachment
  nlapiSendEmail('-5', 'kwolfe@netsuite.com', 'Incoming Transaction', 'Please see attached transaction', null, null, null, file);
}
```

Example 2

In this sample a PDF object is created from a specific statement. This object is then sent as an attachment using **nlapiSendEmail**.

```
function printStatement()
{
  //create an array to set the STATEMENT properties
  var sdate = new Array();
  sdate.startdate = '02/07/2008';
  sdate.statementdate = '03/01/2008';
  sdate.openonly = 'T';

  //print the statement to a PDF file object
```

```
var file1 = nlapiPrintRecord('STATEMENT', 87, 'PDF', sdate);

//send the PDF as an attachment
nlapiSendEmail('-5', 'kwolfe@netsuite.com', 'Regular Statement', 'Please see attached
statement', null, null, null, file1);
}
```

Example 3

This sample shows how to create a PDF of a particular transaction. First the **file** variable is set to a PDF file object. This PDF is then returned as an **nlobjResponse** object. The response object content type is set to PDF (using the **nlobjFile.getType** method). Finally, the output of the response object is written to the server.

```
var file = nlapiPrintRecord('TRANSACTION', 1799, 'PDF', null);
response.setContentType(file.getType());
response.write(file.getValue());
```


[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiSubmitCSVImport(nlobjCSVImport)

Submits a CSV import job to asynchronously import record data into NetSuite. This API can be used to:

- 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.


When the API is executed, the import job is added to the queue. The progress of an import job can be viewed at Setup > Import/Export > View CSV Import Status. For details, see the help topic [Checking CSV Import Status](#).

 **Note:** CSV Imports performed within scripts are subject to the existing application limit of 25,000 records.

Executing this API consumes 100 governance units.

Note that this API cannot be used to 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.

Also note that this API only has access 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. 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 this API. When this API submits a CSV import job based on the saved import map, a single thread and single queue are used.

 **Warning:** This API is only supported for bundle installation scripts, scheduled scripts, and RESTlets. If you attempt to execute this API in another type of script, an error is returned.

Parameters

- **nlobjCSVImport** [required] - [nlobjCSVImport](#) object with methods to set the following: saved import map, primary file, linked file(s) (optional), import job name (optional).

Returns

- On a successful import, an integer that contains the job ID of the import (which is also the identifier for the CSV response file)
- On an unsuccessful import, a string that contains the error message that occurred during validation
This API returns 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.

Since

- Version 2012.2

Examples

This sample uses a script ID to reference the import map and raw string for CSV data:

```
var mappingFileId = "CUSTIMPORTjob1"; //using script id of Saved CSV Import
var primaryFileAsString = "company name, isperson, subsidiary, externalid\ncompanytest001, FALSE, Parent Company, companytest001";

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);
job.setPrimaryFile(primaryFileAsString);
job.setOption("jobName", "job1Import");

//returns the internal id of the new job created in workqueue
var jobId = nlapiSubmitCSVImport(job);
```

This sample uses an internal ID to reference the import map and a CSV file internal ID to reference CSV data:

```
var mappingFileId = 2; //using internal id of Saved CSV Import
var primaryFile = nlapiLoadFile(73); //using the internal id of the file stored in the File Cabinet

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);
job.setPrimaryFile(primaryFile);
job.setOption("jobName", "job2Import");

//returns the internal id of the new job created in workqueue
var jobId = nlapiSubmitCSVImport(job);
```

This sample, which is a multi-file import, uses a script ID to reference the import map and CSV file internal IDs to reference CSV data, and provides a sublist identifier in the [setLinkedFile\(sublist, file\)](#) method:

```
var mappingFileId = "CUSTIMPORTentityMultiFile";

var job = nlapiCreateCSVImport();
job.setMapping(mappingFileId);

//uploaded to File Cabinet <multifile_entity_primary.csv> with internal Id = 73
job.setPrimaryFile(nlapiLoadFile(73));
```

```
//uploaded to File Cabinet <multifile_entity_cust_address.csv> with internal Id = 74
job.setLinkedFile("addressbook", nlapiLoadFile(74));
job.setOption("jobName", "test_ImportMultiFileTransactionRecord-");

var jobId = nlapiSubmitCSVImport(job);
```

For more details about the methods used in these samples, see [nlapiSubmitRecord](#).

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)

Submits and commits new records or changes applied to an existing record and returns the internalId for the committed record. The **nlapiSubmitRecord** function can be used in conjunction with **nlapiCreateRecord** or **nlapiLoadRecord** to create or modify a record related to the current one.

This API is supported in all script types. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.



Important: When using **nlapiSubmitRecord** in a user event script, it is possible that the related record modified or created by the script is committed to the database but the actual record initiating the script fails on save. To avoid this scenario, SuiteScripts that cause actions on records other than the current one should be set to run **afterSubmit**.

Parameters

- **record** {nlobjRecord} [required] - [nlobjRecord](#) object containing the data record
- **doSourcing** {boolean} [optional] - If not set, this argument defaults to false, which means that dependent field values are not sourced. If set to true, sources dependent field information for empty fields. Be aware that **doSourcing** takes the values of **true** or **false**, not T or F. For more information on sourcing, see [Understanding Sourcing in SuiteScript](#) in the NetSuite Help Center.



Important: When working with records in dynamic mode, the value you provide for **doSourcing** will be ignored. Field values will be sourced regardless of whether you set **doSourcing** to true or to false. For information on dynamic scripting, see the help topic [SuiteScript 1.0 Working with Records in Dynamic Mode](#).

- **ignoreMandatoryFields** {boolean} [optional] - Disables mandatory field validation for this submit operation. If set to *true*, ignores all standard and custom fields that were made mandatory through customization. All fields that were made mandatory through company preferences are also ignored.



Important: Use the **ignoreMandatoryFields** 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.

Returns

- An integer value of the committed record's internal ID (for example, 555, 21, or 4).

Throws

- SSS_INVALID_RECORD_OBJ

- SSS_RECORD_OBJ_REQD
- SSS_INVALID_SOURCE_ARG

Example 1

The following example creates an estimate with two items.

```
var record = nlapiCreateRecord('estimate');
record.setFieldValue('entity', 79);
record.setFieldValue('memo', 'Estimate Memo ');

record.setLineItemValue('item', 'item', 1, 21);
record.setLineItemValue('item', 'quantity', 1, 10 );
record.setLineItemValue('item', 'price', 1, 1 );
record.setLineItemValue('item', 'item', 2, 21);
record.setLineItemValue('item', 'quantity', 2, 5 );
record.setLineItemValue('item', 'price', 2, 2 );

var id = nlapiSubmitRecord(record, true);
```

Example 2

Expanding on the Example 2 in `nlapiCreateRecord(type, initializeValues)`, the `createTaskRecord` function now causes a new task record to be created and submitted. This could be tied to an `afterSubmit` function of a user event and deployed to Opportunity records so that each time an Opportunity is created, a task is automatically created.

```
function createTaskRecord()
{
    var taskTitle = 'Follow up regarding new Opportunity';
    var record = nlapiCreateRecord( 'task');
    record.setFieldValue( 'title', taskTitle);
    id = nlapiSubmitRecord(record, true);
}
```

Understanding Sourcing in SuiteScript



Important: If you are working with a record in dynamic mode, the following information does not apply. When submitting a record in dynamic mode, the `doSourcing` argument is ignored. Whether you set `doSourcing` to `true` or to `false`, all field values will be sourced. For information on dynamic scripting, see the help topic [SuiteScript 1.0 Working with Records in Dynamic Mode](#).

When submitting a record in non-dynamic mode, you can retain full control over the data that is written to the system by setting `doSourcing` to `false`, or you can accept sourcing values from NetSuite by setting `doSourcing` to `true`. When set to `true`, fields normally dependent on values from parent fields are automatically pre-populated.

Some advantages to setting `doSourcing` to `true` include:

- Reduces the number of fields that have to be filled out at the same timee retaining data integrity across fields

- Ensures that field values reflect what would normally be submitted when using the entering records via the UI.

Some advantages to setting **doSourcing** to **false** include:

- You retain full control over the data that is written to the system
- Reduces overhead incurred — with **doSourcing** set to **true**, all empty dependent fields on the record (including supported sublists) must be processed

For example, in the UI when a customer is selected on an opportunity record, the leadsource, partner, salesrep, and any custom sourced fields are automatically populated.

If creating an opportunity using SuiteScript with **doSourcing** set to **false**, the leadsource, partner, salesrep, and any custom sourced fields not specifically set by the SuiteScript code would be empty. Therefore, **doSourcing** must be set to **true** for these fields to automatically populate with values based on the value of the customer field.

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiTransformRecord(type, id, transformType, transformValues)

Initializes a new record using data from an existing record of a different type and returns an [nlobjRecord](#). This function can be useful for automated order processing such as creating item fulfillment transactions and invoices off of orders.

This API is supported in client, user event, scheduled, and Suitelet scripts. See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API.

For a list of supported transform types, see [Supported Transformation Types](#).

Parameters

- type** {string} [required] - The record internal ID name. In the NetSuite Help Center, see the help topic [SuiteScript Supported Records](#). The internal ID appears in the column called "Record Internal ID".
- id** {int} [required] - The internalId for the record, for example 555 or 78.
- transformType** {string} [required] - The record internal ID name of the record you are transforming the existing record into
- transformValues** {hashtable} [optional] - An array of field name -> value pairs containing field defaults used for transformation. Note that you can also specify whether you want the record transformation to occur in dynamic mode. For details, see the help topic [SuiteScript 1.0 Working with Records in Dynamic Mode](#).



Important: When doing a sales order to item fulfillment transform on a sales order that has Multiple Shipping Routes (MSR) enabled, you must specify a **shipgroup** value. For example:

```
var itemFulfillment = nlapiTransformRecord('salesorder', id, 'itemfulfillment', { ' shipgroup ' : 5 });
var fulfillmentId = nlapiSubmitRecord(itemFulfillment, true);
```

If you do not specify a value, the system does not know which items on the order are being fulfilled. If a **shipgroup** value is not specified, the value **1** (for the first shipping group) is defaulted. This means that only the items belonging to the first shipping group will be fulfilled when the sales order is transformed. For more information, see the help topic [Multiple Shipping Routes and SuiteScript](#) in the NetSuite Help Center.

Returns

- An `nlobjRecord` object

Throws

- `SSS_INVALID_URL_CATEGORY`
- `SSS_CATEGORY_ARG_REQD`
- `SSS_INVALID_TASK_ID`
- `SSS_TASK_ID_REQD`
- `SSS_INVALID_INTERNAL_ID`
- `SSS_INVALID_EDITMODE_ARG`

Example 1

The following example uses `nlapiTransformRecord` to create an item fulfillment record from an existing sales order.

```
var itemfulfillment = nlapiTransformRecord('salesorder', 1500, 'itemfulfillment');
itemfulfillment.setFieldValue('trandate', nlapiDateToString(new Date()) );
```

Example 2

The next script shows how to create an item receipt from a purchase order.

```
function transformPurchaseOrder()
{
    var fromrecord;
    var fromid;
    var torecord;
    var trecord;
    var qty;

    fromrecord = 'purchaseorder';
    fromid = 26 ; // Transform PO with ID = 26 ;
    torecord = 'itemreceipt';

    // Transform a record with a specific id to a different record type.
    // For example - from PO to Item Receipt
    // Get the object of the transformed record.
    trecord = nlapiTransformRecord(fromrecord, fromid, torecord);
    qty = trecord.getLineItemValue('item', 'quantity', 1 );
    trecord.setLineItemValue('item', 'quantity', 1, '2' );
    var id1 = nlapiSubmitRecord(trecord, true);

    nlapiSendEmail(-5, -5, 'Transform Email' + 'Original Qty = ' + qty + ' ' + 'Record Created = ' + id1 , null);
}
```

Example 3

This script shows how to create an assembly build record from an assembly item, as well as how to set the department field before submitting the new record.

```
function transformAssemblyItem()
{
    var fromRecord = 'assemblyitem';
    var fromId = 328;
    var toRecord = 'assemblybuild';

    var record = nlapiTransformRecord(fromRecord, fromId, toRecord, {'quantity': '1', 'location': '1'});
    record.setFieldValue('department', '1');
    var id = nlapiSubmitRecord(record, false);
}
```

Example 4

The following script shows how to create an assembly build record from an assembly item, as well as how to set the quantity of the member items.



Important: The following sample references the Components (**component**) sublist, which does not yet officially support SuiteScript. This sample is meant for illustrative purposes only. It is meant only to show how to set the values for `nlapiTransformRecord(type, id, transformType, transformValues)`.

```
/* Assembly item name = Computer , Id = 328
2 Member components of Assembly item
Member 1 Name = CPU - Quantity = 2
Member 2 Name = Memory - Quantity = 4
*/

function transformAssemblyItem()
{
    var fromRecord = 'assemblyitem';
    var fromId = 328; // Id of the assembly item
    var toRecord = 'assemblybuild';
    var defaultV = new Array();

    // Default quantity to build
    defaultV.quantity = 1;
    // Default location Id if Multi Location Inventory is enabled.
    defaultV.location = '3';


    var record = nlapiTransformRecord(fromRecord, fromId, toRecord, defaultV);
    // Set quantity of member 1 to 4
    record.setLineItemValue('component', 'quantity', 1, 4);
    // Set quantity of member 2 to 8
    record.setLineItemValue('component', 'quantity', 2, 8);
    var id = nlapiSubmitRecord(record, false);
}
```

Supported Transformation Types

Certain NetSuite record types cannot be created as standalone records. They are always created from another record type because of relationships between the record types. The `nlapiTransformRecord` API can be used to create these types of records.

The following table shows the transformations that are supported in NetSuite:

Record Type	Record Name	Transform Type	Transform Name (Target Record)
assemblyitem	Build/Assembly	assemblybuild	Assembly Build
assemblybuild	Assembly Build	assemblyunbuild	Assembly Unbuild
cashsale	Cash Sale	cashrefund	Cash Sale
customer	Customer	cashsale	Cash Sale
customer	Customer	customerpayment	Customer Payment
customer	Customer	estimate	Quote
customer	Customer	invoice	Invoice
customer	Customer	opportunity	Opportunity
customer	Customer	salesorder	Sales Order
employee	Employee	expensereport	Expense Report
employee	Employee	timebill	Time
estimate	Quote	cashsale	Cash Sale
estimate	Quote	invoice	Invoice
estimate	Quote	salesorder	Sales Order
intercompanytransferorder	Intercompany Transfer Order	itemfulfillment	Item Fulfillment
intercompanytransferorder	Intercompany Transfer Order	itemfulfillment	Item Fulfillment
intercompanytransferorder	Intercompany Transfer Order	itemfulfillment	Item Fulfillment
intercompanytransferorder	Intercompany Transfer Order	itemreceipt	Item Receipt
invoice	Invoice	creditmemo	Credit Memo
invoice	Invoice	customerpayment	Customer Payment
invoice	Invoice	returnauthorization	Return Authorization
lead	Lead	opportunity	Opportunity
opportunity	Opportunity	cashsale	Cash Sale
opportunity	Opportunity	estimate	Quote
opportunity	Opportunity	invoice	Invoice
opportunity	Opportunity	salesorder	Sales Order
prospect	Prospect	estimate	Quote
prospect	Prospect	opportunity	Opportunity
prospect	Prospect	salesorder	Sales Order

Record Type	Record Name	Transform Type	Transform Name (Target Record)
purchaseorder	Purchase Order	itemreceipt	Item Receipt
purchaseorder	Purchase Order	vendorbill	Vendor Bill
purchaseorder	Purchase Order	vendorreturnauthorization	Vendor Return Authorization
returnauthorization	Return Authorization	cashrefund	Cash Refund
returnauthorization	Return Authorization	creditmemo	Credit Memo
returnauthorization	Return Authorization	itemreceipt	Item Receipt
returnauthorization	Return Authorization	revenuecommitmentreversal	Revenue Commitment Reversal
			 Note: The return authorization must be approved and received for this transform to work.
salesorder	Sales Order	cashsale	Cash Sale
salesorder	Sales Order	invoice	Invoice
salesorder	Sales Order	itemfulfillment	Item Fulfillment
salesorder	Sales Order	returnauthorization	Return Authorization
salesorder	Sales Order	revenuecommitment	Revenue Commitment
transferorder	Transfer Order	itemfulfillment	Item Fulfillment
transferorder	Transfer Order	itemreceipt	Item Receipt
vendor	Vendor	purchaseorder	Purchase Order
vendor	Vendor	vendorbill	Vendor Bill
vendorbill	Vendor Bill	vendorcredit	Vendor Credit
vendorbill	Vendor Bill	vendorpayment	Vendor Payment
vendorbill	Vendor Bill	vendorreturnauthorization	Vendor Return Authorization
vendorreturnauthorization	Vendor Return Authorization	itemfulfillment	Item Fulfillment
vendorreturnauthorization	Vendor Return Authorization	vendorcredit	Vendor Credit
workorder	Work Order	assemblybuild	Assembly Build
workorder	Work Order	workorderclose	Work Order Close
workorder	Work Order	workordercompletion	Work Order Completion
workorder	Work Order	workorderissue	Work Order Issue

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlapiVoidTransaction(transactionType, recordId)

When you void a transaction, its total and all its line items are set to zero, but the transaction is not removed from the system. NetSuite supports two types of voids: direct voids and voids by reversing journal. See the help topic [Voiding, Deleting, or Closing Transactions](#) for additional information.

nlapiVoidTransaction voids a transaction and then returns an id that indicates the type of void performed. If a direct void is performed, **nlapiVoidTransaction** returns the ID of the record voided. If a void by reversing journal is performed, **nlapiVoidTransaction** returns the ID of the newly created voiding journal.

The type of void performed depends on the targeted account's preference settings:

- If the Using Reversing Journals preference is disabled, **nlapiVoidTransaction** performs a direct void. See the help topic [Supported Transaction Types — Direct Void](#) for a list of transactions that support direct voids.
- If the Using Reversing Journals preference is enabled, **nlapiVoidTransaction** performs a void by reversing journal. See the help topic [Supported Transaction Types — Void by Reversing Journal](#) for a list of transactions that support voids by reversing journal.



Important: After you successfully void a transaction, you can no longer make changes to the transaction that impact the general ledger

This API is supported in the following script types:

- Client
- User Event
- Scheduled
- Suitelet
- RESTlet
- Workflow Action

The Governance on this API is 10.

Parameters

- transactionType {string} [required] — internal ID of the record type to be voided. See the help topics [Supported Transaction Types — Direct Void](#) and [Supported Transaction Types — Void by Reversing Journal](#) for a list of valid arguments.
- recordId {int} [required] — the internal ID of the specific record to be voided. See the help topic [How do I find a record's internal ID?](#) for additional information.

Returns

- An id that indicates the type of void performed
 - If a direct void is performed, **nlapiVoidTransaction** returns the original recordId value passed in.
 - If a void by reversing journal is performed, **nlapiVoidTransaction** returns the ID of the newly created voiding journal.

Throws

- `INVALID_RCRD_TYPE` — if the `transactionType` argument passed is not valid
- `RCRD_DSNT_EXIST` — if the `recordId` argument passed is not valid
- `THIS_TRANSACTION_HAS_ALREADY_BEEN_VOIDED` — if you attempt to void a transaction that has already been voided
- `VOIDING_REVERSAL_DISALLWD` — if you attempt to void a transaction with inventory impact

Since

- Version 2013 Release 2

Example

The following code creates a new Cash Refund transaction and then voids the transaction. Note that if the Using Reversing Journals preference is enabled, the void will not be successful since Cash Refund is not a supported transaction type for voids by reversing journal. See the help topics [Supported Transaction Types — Direct Void](#) and [Supported Transaction Types — Void by Reversing Journal](#).

```
var recordtype = 'cashrefund';
var rec = nlapiCreateRecord(recordtype);
rec.setFieldValue('entity', 48);
rec.setFieldValue('location', 1);
rec.setFieldValue('paymentmethod', 1);
rec.setLineItemValue('item', 'item', 1, 233);
rec.setLineItemValue('item', 'amount', 1, 150);
var id = nlapiSubmitRecord(rec);

var voidingId;
var errorMsg;
try {
    voidingId = nlapiVoidTransaction(recordtype, id);
}
catch(e) {
    errorMsg = e;
}
```

Supported Transaction Types — Direct Void

Supported Transaction Types for Direct Void	Internal ID
Cash Refund	cashrefund
Cash Sale	cashsale
Credit Memo	creditmemo
Customer Deposit	customerdeposit
Customer Payment	customerpayment
Customer Refund	customerrefund

Supported Transaction Types for Direct Void	Internal ID
Estimate	estimate
Expense Report	expensereport
Intercompany Journal Entry	intercompanyjournalentry
Invoice	invoice
Journal Entry	journalentry
Paycheck Journal	paycheckjournal
Return Authorization	returnauthorization
Sales Order	salesorder
Transfer Order	transferorder
Vendor Bill	vendorbill
Vendor Credit	vendorcredit
Vendor Payment	vendorpayment
Vendor Return Authorization	vendorreturnauthorization
Work Order	workorder

Supported Transaction Types — Void by Reversing Journal

Supported Transaction Types for Void by Reversing Journal	Internal ID
Check	check
Vendor Payment	vendorpayment
Customer Refund	customerrefund
Custom Transaction	customtransaction_nameOfCustomTransactionType For more details, see the help topic Custom Transaction .

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlobjCSVImport

See [nlobjCSVImport](#) - defined in the section on [Standard Objects](#).

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

nlobjRecord

See [nlobjRecord](#) - defined in the section on [Standard Objects](#).

[Back to Record APIs](#) | [Back to SuiteScript Functions](#)

Subrecord APIs

For an overview of NetSuite subrecords, see the help topic [Working with Subrecords in SuiteScript](#).

The subrecord APIs that contain “LineItem” are for creating and working with subrecords from a sublist field on the parent record. The APIs that do not have “LineItem” in the name are for creating and working with subrecords from a body field on the parent record.

Note that most of the functions listed below return an [nlobjSubrecord](#) object. After creating or editing a subrecord, you must save your changes using the [nlobjSubrecord.commit\(\)](#) method. You must then save the subrecord's parent record using [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#). If you do not commit both the subrecord and the parent record, all changes to the subrecord are lost. For complete details on saving subrecords, see the help topic [Saving Subrecords Using SuiteScript](#).

All APIs listed below are in alphabetical order.

- [nlapiCreateCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiCreateSubrecord\(fldname\)](#)
- [nlapiEditCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiEditSubrecord\(fldname\)](#)
- [nlapiRemoveCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiRemoveSubrecord\(fldname\)](#)
- [nlapiViewCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiViewLineItemSubrecord\(sublist, fldname, linenum\)](#)
- [nlapiViewSubrecord\(fldname\)](#)
- [nlobjSubrecord](#)
- [nlobjRecord](#) - all methods with “subrecord” in method signature

nlapiCreateCurrentLineItemSubrecord(sublist, fldname)

Returns a [nlobjSubrecord](#) object. Use this API to create a subrecord from a **sublist** field on the parent record.



Important: This API should only be used in user event scripts on the parent record. Note, however, this API is not supported in **beforeLoad** user event scripts. This API is also not currently supported in form-level or record-level client SuiteScripts associated with the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Creating an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to create a subrecord from a **body** field on the parent record.



Important: This API should only be used in user event scripts on the parent record. Note, however, this API is not supported in **beforeLoad** user event scripts. This API is not supported in form-level or record-level client SuiteScripts deployed on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- **nlobjSubrecord**

Since

- Version 2011.2

Example

See the help topic [Creating an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiEditCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to edit a subrecord from a **sublist** field on the parent record.



Important: This API should only be used in user event scripts on the parent record. This API is not currently supported in form-level or record-level client SuiteScripts associated with the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Editing an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiEditSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to edit a subrecord from a **body** field on the parent record.



Important: This API should only be used in user event scripts on the parent record. This API is not currently supported in form-level or record-level client SuiteScripts deployed on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Editing an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to remove a subrecord from a **sublist** field on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- void

Since

- Version 2011.2

Example

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiRemoveSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to remove a subrecord from a **body** field on the parent record.

This API is currently used only in the context of the / Numbered Inventory feature. For information, see the help topic [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#). Also see the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, *inventorydetail* as the ID for the Inventory Details body field).

Returns

- void

Since

- Version 2011.2

Example

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiViewCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **sublist** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in **read-only** mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this API.

You can call this API when you want your script to read the [nlobjSubrecord](#) object of the current sublist line you are on. After you get the [nlobjSubrecord](#) object, you can use regular record API to access its values.

This API is currently used only in the context of the / Numbered Inventory feature. For information, see the help topic [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#). Also see the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiViewLineItemSubrecord(sublist, fldname, linenum)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **sublist** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in read-only mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this function.

You can call this API when you want to read the value of a line you are **not** currently on (or have not selected). For example, if you are editing line 2 as your current line, you can call **nlapiViewLineItemSubrecord** on line 1 to get the value of line 1.

This API is currently used only in the context of the / Numbered Inventory feature. For information, see the help topic [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#). Also see the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).
- **linenum** {int} [required] - The line number for the sublist field. Note the first line number on a sublist is 1 (not 0).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlapiViewSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **body** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in read-only mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this function.

This API is currently used only in the context of the / Numbered Inventory feature. For information, see the help topic [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#). Also see the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- **nlobjSubrecord**

Since

- Version 2011.2

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlobjSubrecord

See [nlobjSubrecord](#) - defined in the section on [Standard Objects](#).

[Back to Subrecord APIs](#) | [Back to SuiteScript Functions](#)

nlobjRecord

See [nlobjRecord](#) - defined in the section on [Standard Objects](#). If you have used SuiteScript to load the parent record, you will use the “subrecord related” methods on **nlobjRecord** to create and access a subrecord.

Field APIs

For an overview of NetSuite fields, see the help topic [Working with Fields](#).

All APIs listed below are in alphabetical order.

- [nlapiDisableField\(fldnam, val\)](#)

- `nlapiGetField(fldnam)`
- `nlapiGetFieldText(fldnam)`
- `nlapiGetFieldTexts(fldnam)`
- `nlapiGetFieldValue(fldnam)`
- `nlapiGetFieldValues(fldnam)`
- `nlapiInsertSelectOption(fldnam, value, text, selected)`
- `nlapiLookupField(type, id, fields, text)`
- `nlapiRemoveSelectOption(fldnam, value)`
- `nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)`
- `nlapiSetFieldTexts(fldname, txts, firefieldchanged, synchronous)`
- `nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)`
- `nlapiSetFieldValues(fldnam, value, firefieldchanged, synchronous)`
- `nlapiSubmitField(type, id, fields, values, doSourcing)`
- `nlobjField`

nlapiDisableField(fldnam, val)

Sets the field to disabled or enabled based on the value (true or false). This API is supported in client scripts only.

Parameters

- **fldnam** {string} [required] - The internal ID name of the field to enable/disable
- **val** {boolean **true** | **false**} [required] - If set to **true**, the field is disabled. If set to **false**, it is enabled.

Returns

- void

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetField(fldnam)

Use this function to obtain **body** field metadata. Calling this function instantiates the **nlobjField** object, and you can use the methods available to **nlobjField** to get field metadata.

This API is supported in client and user event scripts only.

Note: When **nlapiGetField** is used in **client scripts**, the field object returned is **read-only**. This means that you can use **nlobjField** getter methods in client scripts (to obtain metadata about the field), but you cannot use **nlobjField** setter methods to set field properties. There is one exception, **nlobjField.setDisplayType** can be used to set the display type.

Note: To obtain metadata for sublist fields, use **nlapiGetLineItemField(type, fldnam, linenum)**.

Parameters

- **fldnam** {string} [required] - The internal ID of the field.

Returns

- Returns an [nlobjField](#) object representing this field

Since

- Version 2009.1

Example


The following script is attached to a Sales Order. The `nlapiGetField` API returns a `nlobjField` object. This script then uses the field object methods `getType` and `getLabel` to return the field's type and UI label.

```
function clientScript(type)
{
    var field = nlapiGetField('memo'); // specify the internalId of the Memo field
    alert(field.getType()); // returns text as the field type
    alert(field.getLabel()); // returns Memo as the field UI label
}
```

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetFieldText(fldnam)

Use this API to get the text value (rather than the internal ID value) of a field. This API is available in client and user event scripts only.


 **Note:** This API is not supported on subrecords.

Parameters

- `fldnam` {string} [required] - The internal ID of the field.

Returns

- The string UI display name for a select field corresponding to the current selection

 **Note:** For multiselect fields, this API returns the display names as a string with the `\u0005` separator.

Example

The following client script reads the text value of the Department field. If the Department field contains no value when the page loads, an alert is thrown telling users to select the Service department (one of the text values in the Department dropdown field). If the page loads and the department field defaults to Sales, an alert is thrown telling users to select the Service department instead.

```
function pageInit_getFieldTextTest() {
    var departId = nlapiGetFieldText('department');

    if (departId == '') {
        alert('Please specify the Service department');
    }
}
```

```

}

if (departId == 'Sales') {
  alert('Please select the Service department');
}
}

```

The screenshot shows the NetSuite 'Sales Order' form. A JavaScript alert dialog box is displayed in the center, titled 'Message from webpage', with a warning icon and the message 'Please specify the Service department'. The dialog has an 'OK' button. In the background, the 'Sales Order' form is visible, with the 'DEPARTMENT' field in the 'Classification' section highlighted by a red rectangle. The form includes sections for Primary Information, Sales Information, Classification, and Intercompany Management. A summary table on the right shows various costs and totals.

Summary	
TOTAL (ALT. SALES)	0.00
SUBTOTAL	0.00
DISCOUNT ITEM	0.00
SHIPPING COST	
ROLLING COST	
CERTIFICATE	0.00
L	0.00

Important: `nlapigetFieldText` cannot be used on **hidden** fields or non select fields.

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

`nlapigetFieldTexts(fldnam)`

Returns the display names for a multiselect field corresponding to the current selection. This API is available in client and user event scripts only.

Parameters

- **fldnam** {string} [required] - The internal ID of the field whose display values are returned

Returns

- The display names for a multiselect field as an Array.

Since

- Version 2009.1

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetFieldValue(fldnam)

Use this function to get the internal ID of a field. For example, if the customer Abe Simpson appears in a field, this function will return 87, which represents the internal ID value of the Abe Simpson customer record. Note that if you are getting the value of an inline check box, the return value will be **F** if the field is unset.

This API is available in client and user event scripts only, and it not supported on subrecords.

Also be aware that this API is not supported during delete events. Calling **nlapiGetFieldValue** on a record you are attempting to delete will return a user error.

Also note that if you are trying to return an array of values from a multiselect field, it is recommended that you use the [nlapiGetFieldValues\(fldnam\)](#) API.

Finally, NetSuite recommends that you read the topic [Getting Field Values in SuiteScript](#), which addresses the rare instances in which the value returned by this API is inconsistent.

Parameters

- **fldnam** {string} [required] - The internal ID of the field.

Returns

- The string value of a field on the **current** record, or returns *null* if the field does not exist on the record or the field is restricted.



Important: If you choose to use **nlapiGetFieldValue(fldnam)** to return values from a multiselect field (rather than use the [nlapiGetFieldValues\(fldnam\)](#) API), you must delimit multiselect values using CHR(5) or the ANSI control character with code 5.

For example:

```
function stringToArray (str)
{
    //Use ChrCode 5 as a separator
    var strChar5 = String.fromCharCode(5);

    //Use the Split method to create an array,
    //where ChrCode 5 is the separator/delimiter
    var multiSelectStringArray = str.split(strChar5);

    return multiSelectStringArray;
}

function displayResult ()
{
    var str = stringToArray(nlapiGetFieldValue('custentity8'));
    alert (str);
}
```

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetFieldValues(fldnam)

Use this function to get an array of internal ID values for a multiselect field.

This API is available in client and user event scripts only.

Parameters

- **fldnam** {string} [required] - The internal ID of the field. For a list of fields that are supported in SuiteScript and their internal IDs, see the *SuiteScript Reference Guide*.

Returns

- The values of a multiselect field as an Array on the current record. Returns **null** if the field does not exist on the record or the field is restricted.

Since

- Version 2009.1

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiInsertSelectOption(fldnam, value, text, selected)

Adds a select option to a select/multiselect field added via script. Note that this API can only be used on select/multiselect fields that are added via the [UI Objects](#) API (for example, in Suitelets or beforeLoad user events scripts).



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with **nlapiInsertSelectOption**. The select values are determined by the source record or list.

Parameters

- **fldnam** {string} [required] - The internalId of the scripted field
- **value** {string | int} [required] - A unique value for the select option. Note that the datatype for this argument will vary depending on the value that is set. For example, you may assign numerical values such as 1, 2, 3 or string values such as option1, option2, option3.
- **text** {string} [required] - The display name of the select option
- **selected** {boolean **true** | **false**} [optional] - If not set, this argument defaults to false. If set to **true**, the select option becomes the default option.

Returns

- void

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiLookupField(type, id, fields, text)

Performs a search for one or more **body** fields on a record. This function supports joined-field lookups. Note that the notation for joined fields is: **join_id.field_name**

Note: In search/lookup operations, custom fields of type "long text" are truncated. In accounts with multiple languages enabled, the returned value is truncated at 1,300 characters. In accounts that don't use multiple languages, the field return truncates at 3,900 characters.

See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API. This API is available in client, user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **type** {string} [required] - The record internal ID name. In the NetSuite Help Center, see the help topic [SuiteScript Supported Records](#). Record IDs are listed in the "Record Internal ID" column.
- **id** {int} [required] - The internalId for the record, for example 777 or 87.
- **fields** {string | string[]} [required] - Sets an array of column/field names to look up, or a single column/field name. The **fields** parameter can also be set to reference joined fields (see the third code sample).
- **text** {boolean} [optional] - If not set, this argument defaults to false and the internal ID of the dropdown field is returned. If set to **true**, this argument returns the UI display name for this field or fields (valid only for SELECT | IMAGE | DOCUMENT fields).

Returns

- {string | hashtable} - A single value (or text) or an associative Array of field name -> value (or text) pairs depending on the field's argument.

Example 1

The following example is executed from an Opportunity afterSubmit User Event script and returns salesrep detail information.

```
var record = nlapiGetNewRecord();
var salesrep = record.getFieldValue('salesrep')
var salesrep_email = nlapiLookupField('employee', salesrep, 'email');
var salesrep_supervisor = nlapiLookupField('employee', salesrep, 'supervisor', true);
```

Example 2

The following example shows how to use the **nlapiLookupField** function to return an array of field names. In this example, the email, phone, and name fields are returned from a Customer record.

```
var fields = ['email', 'phone', 'entityid']
var columns = nlapiLookupField('customer', customer_id, fields);
var email = columns.email
var phone = columns.phone
var name = columns.entityid
```

Example 3

The following example returns the partner phone number for a customer specified by the customer recordId. In this scenario, using a joined field lookup eliminates having to perform two different **nlapiLookupField** calls (one for **customer.partner** and another for **partner.phone**) to obtain the same information.

```
nlapiLookupField('customer', customer_id, 'partner.phone')
```

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiRemoveSelectOption(fldnam, value)

Removes a single select option from a select or multiselect field added via script. Note that this API call can only be used on select/multiselect fields that are added via the [UI Objects](#) API (for example on Suitelets or beforeLoad user event scripts).

Parameters

- **fldnam** {string} - The name of the scripted field
- **value** {string} - The value of the select option to be removed or null to delete all the options

Returns

- void

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)


nlapiSetFieldText(fldname, txt, firefieldchanged, synchronous)

Sets the value of a select field on the current record using the UI display name. This API can be used in user event **beforeLoad** scripts to initialize a field on new records or to initialize a non-stored field. (Non-stored fields are those that have the **Store Value** preference unchecked on the custom field page.)

This function is available in client and user event scripts only.


Parameters

- **fldname** {string} [required] - The name of the field being set
- **txt** {string} [required] - The display name associated with the value that the field is being set to
- **firefieldchanged** {boolean} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

 **Note:** The **firefieldchanged** parameter takes the values of **true** or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as **true**.

In client scripts, if you do not set the value of **synchronous**, the default value is **false**, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

 **Note:** In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)


nlapiSetFieldTexts (fldname, txts, firefieldchanged, synchronous)

Sets the values of a multi-select field on the current record using the UI display names. This API can be used in user event **beforeLoad** scripts to initialize a field on new records or to initialize a non-stored field. (Non-stored fields are those that have the **Store Value** preference unchecked on the custom field page.)

This function is available in client and user event scripts only.


Parameters

- **fldname** {string} [required] - The name of the field being set
- **txts** {string[]} [required] - The display names associated with the values that the field is being set to
- **firefieldchanged** {boolean} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

 **Note:** The **firefieldchanged** parameter takes the values of **true** or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

 **Note:** In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

Since

- 2009.1

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)


Sets the value of a body field. This API can be used in user event **beforeLoad** scripts to initialize a field on new records or to initialize a non-stored field. (Non-stored fields are those that have the **Store Value** preference unchecked on the custom field page.)

For client-side scripting, this API can be triggered by a **PageInit** client event trigger.


This API is available in client and user event scripts only.

Parameters

- **fldnam** {string} [required] - The internal ID name of the field being set
- **value** {string} [required] - The value the field is being set to.


 **Note:** Check box fields take the values of T or F, not **true** or **false**

- **firefieldchanged** {boolean} [optional] - If *true*, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

 **Note:** The **firefieldchanged** parameter takes the values of **true** or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

 **Note:** In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

Example

This sample shows the relationship between setting the value for a parent field and the sourcing that occurs synchronously for a child field.

In this example the value for the Customer (**entity**) field gets set to a specific customer when a Sales Order first loads. After the value is set for **entity**, the value of the Sales Rep (**salesrep**) field synchronously sources, and an alert is thrown to identify the Sales Rep. If the value of the **synchronous** parameter had not been set to **true** for *nlapiSetFieldValue*, there is a possibility that the alert would be thrown **before** it included the sales rep ID. With **synchronous** set to **true**, the alert cannot be thrown **until** the **salesrep** field data has been correctly sourced from the **entity** field.

```
//Set this script to run on a PageInit (page load) client event trigger
function setCustomer()
{
    nlapiSetFieldValue('entity', 87, null, true);
}

//Set this script to run on a FieldChanged client trigger. The Sales Rep
//(salesrep) field sources its data based on the value of the entity field.
function setSalesRep(type, fld)
{
    if (fld == 'entity')
    {
        var val = nlapiGetFieldValue('salesrep');
        alert('sales rep is ' + val);
    }
}
```

}

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetFieldValues (fldnam, value, firefieldchanged, synchronous)

Sets the value of a multiselect body field on a current record. This API can be used for user event **beforeLoad** scripts to initialize fields on new records or non-stored fields. (Non-stored fields are those that have the **Store Value** preference unchecked on the custom field page.

For client-side scripting, this API can be triggered by a **PageInit** client event trigger.

This API is available in client and user event scripts only.

Parameters

- **fldnam** {string} [required] - The internal ID name of the field being set
- **value** {string} [required] - The value the field is being set to (Array).
- **firefieldchanged** {boolean **true** || **false**} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**.



Important: This parameter is available in client scripts only. See [Using the Fire Field Changed Parameter](#) for more information.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as **true**.

In client scripts, if you do not set the value of **synchronous**, the default value is **false**, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.



Important: In client scripts, the **synchronous** parameter takes the value of **true** or **false**, not T or F.

Returns

- void

Since

- 2009.1

Example

```
var values = new Array() // define a new Array and set customers
values[0] = '80'; // 80 references the internal ID of first customer, Abe Simpson
values[1] = '81'; // 81 references the internal ID of the second customer, Abe Lincoln

// set values for the multiselect field called Customers Multiselect Field
nlapiSetFieldValues('custbody23', values);
```

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlapiSubmitField(type, id, fields, values, doSourcing)

Updates one or more body fields or custom fields on a record. This function can be used on any record that supports inline editing and on any body field or custom field that supports inline editing. Note that this function cannot be used to update sublist “line item” fields.

The `nlapiSubmitField` function is a companion function to `nlapiLookupField(type, id, fields, text)`.

`nlapiSubmitField` is available in client, user event, scheduled, portlet, and Suitelet scripts.

See the help topic [SuiteScript 1.0 API Governance](#) for the unit cost associated with this API. Note that the metering for this API is on a per-call basis, not per updated line. For example you can update five fields with one call to `nlapiSubmitField`, and the entire operation will cost 10 units (if the API is executing on a standard transaction record).



Important: In the NetSuite UI, users cannot set fields that are not inline editable. SuiteScript, however, **does** let you set non inline editable fields using `nlapiSubmitField`, but this is NOT the intended use for this API. See the help topic [Consequences of Using nlapiSubmitField on Non Inline Editable Fields](#) to learn about the increased governance cost of using this API on non inline editable fields.

Parameters

- **type** {string} [required] - The record internal ID name of the record you are updating.
- **id** {int} [required] - The internalId for the record, for example 777 or 87
- **fields** {string | string[]} [required] - An Array of field names being updated -or- a single field name
- **values** {string | string[]} [required] - An Array of field values being updated -or- a single field value
- **doSourcing** {boolean **true** | **false**} [optional] - If not set, this argument defaults to false and field sourcing does not occur. If set to true, sources in dependent field information for empty fields.

Returns

- void

Example 1

The following example inactivates a set of custom records returned by a saved search. Note that the **Inactive** field on the Custom Record definition page is check box. In SuiteScript, check boxes always take the value or T or F, not true or false.

```
var records = nlapiSearchRecord('customrecord_oldrecords', 'customsearch_records_to_inactivate');
for ( var i = 0; i < records.length; i++ ) {
    nlapiSubmitField(records[i].getRecordType(), records[i].getId(), 'isinactive', 'T');
}
```

Example 2

This sample shows `nlapSubmitField` in the context of a Suitelet.

```
function updateFields(request, response) {
  //item fulfillment
  nlapSubmitField('itemfulfillment', 55, 'memo', 'Memo for item fulfillment', true);

  //customer
  nlapSubmitField('customer', 87, 'comments', 'Enter custom memo here', true);
}
```

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

nlobjField

See [nlobjField](#) - defined in the section on [UI Objects](#).

[Back to Field APIs](#) | [Back to SuiteScript Functions](#)

Sublist APIs

For an overview of NetSuite sublists, see the help topic [Working with Subtabs and Sublists](#).

All APIs listed below are in alphabetical order.

- [nlapiCancelLineItem\(type\)](#)
- [nlapiCommitLineItem\(type\)](#)
- [nlapiDisableLineItemField\(type, fldnam, val\)](#)
- [nlapiFindLineItemMatrixValue\(type, fldnam, val, column\)](#)
- [nlapiFindLineItemValue\(type, fldnam, val\)](#)
- [nlapiGetCurrentLineItemIndex\(type\)](#)
- [nlapiGetCurrentLineItemMatrixValue\(type, fldnam, column\)](#)
- [nlapiGetCurrentLineItemText\(type, fldnam\)](#)
- [nlapiGetCurrentLineItemValue\(type, fldnam\)](#)
- [nlapiGetLineItemCount\(type\)](#)
- [nlapiGetLineItemField\(type, fldnam, linenum\)](#)
- [nlapiGetLineItemMatrixField\(type, fldnam, linenum, column\)](#)
- [nlapiGetLineItemMatrixValue\(type, fldnam, linenum, column\)](#)
- [nlapiGetLineItemText\(type, fldnam, linenum\)](#)
- [nlapiGetLineItemValue\(type, fldnam, linenum\)](#)
- [nlapiGetMatrixCount\(type, fldnam\)](#)
- [nlapiGetMatrixField\(type, fldnam, column\)](#)
- [nlapiGetMatrixValue\(type, fldnam, column\)](#)
- [nlapiInsertLineItem\(type, line\)](#)
- [nlapiInsertLineItemOption\(type, fldnam, value, text, selected\)](#)
- [nlapiIsLineItemChanged\(type\)](#)

- `nlapiRefreshLineItems(type)`
- `nlapiRemoveLineItem(type, line)`
- `nlapiRemoveLineItemOption(type, fldnam, value)`
- `nlapiSelectLineItem(type, linenum)`
- `nlapiSelectNewLineItem(type)`
- `nlapiSetCurrentLineItemMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)`
- `nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)`
- `nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)`
- `nlapiSetCurrentLineItemValues(type, fldnam, values, firefieldchanged, synchronous)`
- `nlapiSetLineItemValue(type, fldnam, linenum, value)`
- `nlapiSetMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)`
- `nlobjSubList`

nlapiCancelLineItem(type)

Cancels any uncommitted changes to the current line of a sublist

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiCommitLineItem(type)

Saves/commits the changes to the current line in a sublist. This is the equivalent of clicking **Done** for a line item in the UI.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiDisableLineItemField(type, fldnam, val)

Sets the line item field of a sublist to disabled or enabled based on the value (true or false). This function is only supported in client scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the line item field to enable/disable
- **val** {boolean **true** | **false**} [required] - If set to **true** the field is disabled. If set to **false** it is enabled.


Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiFindLineItemMatrixValue(type, fldnam, val, column)

This API returns the line number of a particular price in a column. If the value is present on multiple lines, it will return the line item of the first line that contains the value. This API is supported in client and user event scripts. Use this API on a matrix sublists only.

 **Note:** Currently the Pricing sublist and Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript. For details, see the help topics [Pricing Sublist / Pricing Matrix](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field
- **val** {string} [required] - The value of the field
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.

Returns

- The line number (as an integer) of a specified matrix field

Since

- Version 2009.2

Example


This sample shows how to return the line number of a particular price in a column. Note that if the specified value is present on multiple lines, this API returns the line number of the first line that contains the value.

```
var column1 = nlapiFindLineItemMatrixValue('price', 'price', 213.00, 1);
alert('The line number of price 213 from column 1 is. ' + column1);
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiFindLineItemValue(type, fldnam, val)

Use this API to find the line number of a specific field in a sublist. This API can be used on any sublists that supports SuiteScript. This API is supported in client and user event scripts only.

 **Note:** This API is not supported on subrecords.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The field internal ID
- **val** {string} [required] - The value of the field

Returns

- The line number (as an integer) of a specific sublist field

Since

- Version 2009.2


Example

```
nlapiFindLineItemValue('item', 'quantity', '1');
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetCurrentLineItemIndex(type)

Returns the line number of the currently selected line in a group.

 **Note:** The first line number on a sublist is **1** (not 0).

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- The integer value for the currently selected line number in a sublist

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)

Use this API to get the value of the currently selected matrix field. This API should be used on matrix sublists only. This API is supported in client and user event scripts.



Important: Currently the Pricing sublist and Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript. For details, see the help topics [Pricing Sublist / Pricing Matrix](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field being set.
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.

Returns

- The string value of a field on the currently selected line in a matrix sublist. Returns *null* if the field does not exist.

Since

- Version 2009.2

Example

This sample executes on a **pageInit** client event. The script throws an alert to let the user know the values that appear in the first column and the second column of a Pricing sublist.

```
function getCurrentLine()
{
    //Get values for column 1 and column 2
    var column1 = nlapiGetCurrentLineItemMatrixValue('price', 'price', 1);
    var column2 = nlapiGetCurrentLineItemMatrixValue('price', 'price', 2);
    alert('The values in column 1 and 2 are ' + column1 + ' '+column2);
}
```

Example 2

This sample executes on a **validateField** client event. It runs in an account that has the Multiple Currencies feature enabled. The script gets the value specified in the second column of the pricing matrix that appears on the USA currency tab (price1). Based on the value, it then sets values on the British Pound tab (price2). To set line item values, notice the pattern of selecting the line, then setting values, then committing the changes.

```
function validateFieldOnItem(type, fld, column)
{
    if(type == 'price1')
    {
        if(nlapiGetCurrentLineItemMatrixValue('price1', 'price', 1)=='44.00')
        {
            nlapiSetFieldValue('department', 5);
            nlapiSelectLineItem('price2', '1');
            nlapiSetCurrentLineItemMatrixValue('price2', 'price', 1, '11');
            nlapiSetCurrentLineItemMatrixValue('price2', 'price', 2, '12');
            nlapiCommitLineItem('price2');
        }
    }
}
```

```


    }
    return true;
}

```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetCurrentLineItemText(type, fldnam)

Returns the display name (the UI label) of a select field (based on its current selection) on the **currently** selected line. Typically used in validate line functions.


 **Note:** This API is not supported on subrecords.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set

Returns


- The string display name of a select field (based on its current selection) on the currently selected line. Returns **null** if the field does not exist.

 **Note:** For multiselect fields, this API returns the display names as a string with the \u0005 separator.

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetCurrentLineItemValue(type, fldnam)

Returns the value of a sublist field on the currently selected line.

 **Note:** This API is not supported on subrecords.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set

Returns

- The string value of a field on the currently selected line. Returns **null** if field does not exist.

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetCurrentLineItemValues(type, fldnam)

Returns the values of a multiselect sublist field on the currently selected line. One example of a multiselect sublist field is the Serial Numbers field on the Items sublist.

This function is not supported in client SuiteScript. It is meant to be used in user event scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the multiselect field.

Returns

- An array of string values for the multiselect sublist field (on the currently selected line)

Since


- Version 2012.1

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemCount(type)

Use this API to determine the number of line items on a sublist. You can then use APIs such as **nlapiInsertLineItem** or **nlapiRemoveLineItem** to add or remove lines before/after existing lines.

The **nlapiGetLineItemCount** API is available in Client and User Event scripts only. If you want to get the line count of a sublist in a Suitelet, see [nlobjSubList.getLineItemCount\(\)](#).


 **Important:** The first line number on a sublist is **1** (not 0).

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- The integer value for the number of lines in a sublist for the current record

 **Note:** For performance reasons, the return value of this API call is undefined for sublists with custom child records in CSV Import and SOAP web services.

Example

The following sample shows how to use **nlapiGetLineItemCount** to programmatically determine the number of line items on a sublist.

```
function getLineCount()
```

```


{
  var lineNum = nlapiGetLineItemCount('solutions');
  alert('The line item count for this sublist is: ' + lineNum);
}

```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemField(type, fldnam, linenum)

Use this function to obtain **sublist** (line item) field metadata. Calling this function instantiates the **nlobjField** object, and you can use all the methods available to **nlobjField** to get field metadata.

 **Note:** To obtain metadata for body fields, use [nlapiGetField\(fldnam\)](#).

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the sublist field
- **linenum** {int} [optional] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- An [nlobjField](#) object representing this line item field

Since

- Version 2009.1

Example

The following script is attached to a Sales Order. The **nlapiGetLineItemField** API returns a **nlobjField** object. This script then uses the field object methods **getType** and **getLabel** to return the sublist field's type and UI label.

```

function clientSideScript(type, form)
{
  var field = nlapiGetLineItemField('item', 'quantity', 3);
  alert(field.getType()); // returns float as the field type
  alert(field.getLabel()); // returns Quantity as the field UI label
}

```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemMatrixField(type, fldnam, linenum, column)

Use this API to obtain metadata for a field that appears in a matrix sublist. This API is supported in client and user event scripts.

Note: Currently the Pricing sublist and Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript. For details, see the help topics [Pricing Sublist / Pricing Matrix](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

Calling this function instantiates the `nlobjField` object, and you can use all the methods available to the `nlobjField` object.

Note: To obtain metadata for body fields, use `nlapiGetField(fldnam)`.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the field (line) whose value you want returned.
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.

Returns

- An `nlobjField` object representing this sublist field. Returns **null** if the field you have specified does not exist.

Since

- Version 2009.2

Example

This script executes on a `pageInit` client event. It gets the metadata of a matrix field on the Pricing sublist.

```
function getFieldInfo()
{
    var matrixField = nlapiGetLineItemMatrixField('price1', 'price', '1', '1');
    var fieldLabel = matrixField.getLabel();
    var fieldName = matrixField.getName();
    var fieldType = matrixField.getType();
    var fieldMetaInfo = 'Label: '+fieldLabel+' Name: '+fieldName+' Type: '+fieldType ;
    alert('price field metadata is : '+ fieldMetaInfo);
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)

Use this API to get the value of a matrix field that appears on a specific line in a specific column. This API can be used only in the context of a matrix sublist. This API is supported in client and user event scripts.



Important: Currently the Pricing sublist and Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript. For details, see the help topics [Pricing Sublist / Pricing Matrix](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field whose value you want returned.
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- The string value of the matrix field.

Since

- Version 2009.2

Example

This sample executes on a **pageInit** client event. The script will throw an alert that lists the values appearing in columns 1 and 2 on line 1 of the Pricing sublist.

```
function getMatValues()
{
    nlapiSelectLineItem('price', 1);
    var column1 = nlapiGetLineItemMatrixValue('price', 'price', 1, 1);
    var column2 = nlapiGetLineItemMatrixValue('price', 'price', 1, 2);
    alert('Values from row 1 and 2 are ' + column1 + ' ' + column2);
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemText(type, fldnam, linenum)

Returns the display name of a select field (based on its current selection) in a sublist.



Note: This API is not supported on subrecords.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of the display name of a select field (based on its current selection) in a sublist. Returns **null** if field does not exist on the record or the field is restricted.

Note: For multiselect fields, this API returns the display names as a string with the \u0005 separator.

Example

This is a client script that throws an alert with the value of the **myText** variable.

```
function testGetText()
{
    var myText = nlapiGetLineItemText('item', 'item', 1);
    if (myText != '' || myText != null)
    {
        alert ('value obtained is ' +myText);
    }
    else
    {
        alert('value obtained is not valid');
    }
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemValue(type, fldnam, linenum)

Available only in client and user event SuiteScripts. Note that you cannot set default line item values when the line is not in edit mode.

Also, NetSuite recommends that you read the topic [Getting Field Values in SuiteScript](#), which addresses the rare instances in which the value returned by this API is inconsistent.

Note: This API is not supported on subrecords.

Note: Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the field (line item) whose value is being returned
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of a sublist line item

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLineItemValues(type, fldname, linenum)

Returns the values of a multiselect sublist field on a selected line. One example of a multiselect sublist field is the Serial Numbers field on the Items sublist.

This function is not supported in client SuiteScript. It is meant to be used in user event scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the multiselect field
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- An array of string values for the multiselect sublist field

Since

- Version 2012.1

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetMatrixCount(type, fldnam)

Use this API in a matrix sublist to get the number of columns for a specific matrix field. This API is supported in client and user event scripts.

Note: Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API typically would not be used for the Demand Plan Detail sublist. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Note: The first column in a matrix is 1, not 0.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The field internal ID of the matrix field.

Returns

- The integer value for the number of columns of a specified matrix field

Since

- Version 2009.2

Example

This sample executes on a `pageInit` client event. If there are 2 columns in the pricing matrix, the value of 2 will be passed to `matrixCount` variable. If there are 3 columns, the value of 3 will be passed. Note that the **type** parameter is set to **price1**. This means that the Multiple Currencies feature has been enabled in the user's account, and the user is scripting to the USA tab on the Pricing Sublist.

```
function getCount()
{
    var matrixCount = nlapiGetMatrixCount('price1', 'price');
    alert('Matrix Count is ' + matrixCount);
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetMatrixField(type, fldnam, column)

Use this API to get field metadata for a matrix “header” field in a matrix sublist.

Note: Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API is used only for the Pricing sublist. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

For example, if the Quantity Pricing feature is enabled in your account, you will see the **Qty** fields at the top of the pricing matrix. The Qty fields are considered to be the header fields in the pricing matrix. For more information on matrix header fields, see the help topic [Matrix APIs](#) in the NetSuite Help Center.

This API is supported in client and user event scripts.

Parameters

- type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- fldnam** {string} [required] - The internal ID of the matrix header field.
- column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- nlobjField** object

Since

- Version 2009.2

Example

This sample executes on a `pageInit` client event to get the metadata of a matrix header field. In this case, **Qty** is the matrix header field on the Pricing sublist. After you call **nlapiGetMatrixField** you can use all the methods on the **nlobjField** object to get whatever field metadata you might need.

```
function getMatrixHeaderInfo()
{
    var qtyObject = nlapiGetMatrixField('price', 'price', 2);

    var fieldLabel = qtyObject.getLabel();
    var fieldName = qtyObject.getName();
    var fieldType = qtyObject.getType();

    var fieldMetaInfo = 'Label: '+fieldLabel+' Name: '+fieldName+' Type: '+fieldType ;
    alert('Get Quantity Field Meta data ' + fieldMetaInfo);
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetMatrixValue(type, fldnam, column)

Use this API to get the value of a matrix “header” field in a matrix sublist.

Note: Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API is used only for the Pricing sublist. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

For example, if the Quantity Pricing feature is enabled in your account, you will see the **Qty** fields at the top of the pricing matrix. The Qty fields are considered to be the header fields in the pricing matrix. See the help topic [Matrix APIs](#) in the NetSuite Help Center for more information on matrix header fields.

This API is supported in client and user event scripts.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix header field.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- The integer value of a matrix header field. For example, on the Pricing sublist the value of a specified quantity level (Qty) field is returned.

Since

- Version 2009.2

Example 1

This sample executes on a pageInit client event to get the value of the **quantity** level that appears on the second column of the Pricing sublist. Note that the **type** parameter is set to **price1**. This means that the Multiple Currencies feature has been enabled in the user's account, and the user is scripting to the USA tab on the Pricing Sublist.

```
function getMatValue()
{
```

```
var matrixValue = nlapiGetMatrixValue('price1', 'price', 2);
alert('Value in the column is ' + matrixValue);
}
```

Example 2

This sample executes on a `validateField` client event. It gets the value of a quantity (Qty) matrix header field.

```
function validateFieldOnItem(type, fld, column)
{
  if( type=='price1' )
  {
    if(nlapiGetMatrixValue('price1', 'price', '2')=='100')
    {
      alert('Item is available to ship');
      nlapiSetFieldValue('department', 5);
      nlapiSelectLineItem('price2', '1');
      nlapiSetCurrentLineItemMatrixValue('price2', 'price', 1, '100');
      nlapiSetCurrentLineItemMatrixValue('price2', 'price', 2, '90');
      nlapiCommitLineItem('price2');
    }
  }
  return true;
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiInsertLineItem(type, line)

Inserts a line above the currently selected line in a sublist. Available to client and user event scripts only.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **line** {int} [required] - The line number in which to insert new line. Note the first line number on a sublist is **1** (not 0).

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiInsertLineItemOption(type, fldnam, value, text, selected)

Adds a select option to a select/multiselect field that was added through scripting. This field will appear as a line item on a sublist.

Note that this API can only be used on select/multiselect fields that are added via the [UI Objects](#) API (for example on Suitelets or beforeLoad user events).

For performance reasons, you should disable the dropdown list before adding multiple options, then enable the dropdown list when finished.



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with `nlapiInsertSelectOption`. The select values are determined by the source record or list.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the scripted field
- **value** {string | int} [required] - A unique value for the select option. Note that the datatype for this argument will vary depending on the value that is set. For example, you may assign numerical values such as 1, 2, 3 or string values such as option1, option2, option3.
- **text** {string} [required] - The display name of the select option
- **selected** {boolean **true** | **false**} [optional] - If not set, this argument defaults to false. If set to true, the selected option will become the default selection.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiIsLineItemChanged(type)

Determines whether any changes have been made to a sublist.

This API can only be used in client scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- Returns *true* if the currently selected line of the sublist has been edited

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiRefreshLineItems(type)

Makes a server call to refresh staticlist (read-only) sublists. For **inlineeditor** or **editor** sublists, it simply redraws the sublist. This API does not do anything for sublists of type *list*.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiRemoveLineItem(type, line)

Removes the currently selected line in a sublist. Supported in client scripts, user event scripts, and Suitelets.



Important: For user event scripts and Suitelets, you must use the **line** parameter to select the line item. For client scripts, you can use [nlapiSelectLineItem\(type, linenum\)](#).



Note: For Scheduled scripts, use the equivalent record-level method: `nlobjRecord.removeLineItem(group, linenum, ignoreRecalc)`.

Parameters:

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **line** {int} [required for user event scripts and Suitelets] - The line number you want to remove. Note the first line number on a sublist is **1** (not 0).

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiRemoveLineItemOption(type, fldnam, value)

Removes a single select option from a select or multiselect line item field added through a script

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the scripted field.
- **value** {string} [required] - The value of the select option to be removed or **null** to delete all the options.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectLineItem(type, linenum)

Selects an existing line in a sublist

Parameters

- **type** - {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **linenum** - {int} [required] - The line number to select. Note the first line number on a sublist is **1** (not 0).

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectNewLineItem(type)

Use this function if you want to set a value on a sublist line that does not currently exist. This API is the UI equivalent of clicking a sublist tab (for example the Items sublist tab) so that you can then add a new line (or item, in this example) to the sublist.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- void

Example

```
function sampleClientPageInit()
{
    nlapiSetFieldValue('entity', '294');
    // this is the equivalent of selecting the Items sublist tab. You must do this when you want to
    // add new lines to a sublist
    nlapiSelectNewLineItem('item');

    // set the item and location values on the currently selected line
    nlapiSetCurrentLineItemValue('item', 'item', 380, true, true);
    nlapiSetCurrentLineItemValue('item', 'location', 102, true, true);

    // commit the line to the database
    nlapiCommitLineItem('item');
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetCurrentLineItemMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)

This API is typically used in validate line functions to set the value of a matrix sublist field before it has been added to the form. This API is supported in client and user event scripts. Also note that it should be used on matrix sublists only.

Note: Currently the Pricing sublist and Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript. For details, see the help topics [Pricing Sublist / Pricing Matrix](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).
- **value** {string | int} [required] - The value the field is being set to.
- **firefieldchanged** {boolean} [optional] - If **true**, then the field change script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

Note: The **firefieldchanged** parameter takes the values of **true** or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

Note: In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

Since

- Version 2009.2

Example

The following sample is a user event script that executes on a beforeLoad event. This script is set to execute on the Pricing sublist on an Inventory Item record. On the Pricing sublist it will set the Base Price for the first two columns of the USA tab. The presence of the USA tab indicates that the Multiple

Currencies feature is enabled in this account. Therefore, the internal ID of the type parameter in all matrix APIs will be price1.

```
function beforeLoad(type, form)
{
    nlapiSetFieldValue('itemid', '124');

    //Set the pricing matrix header field ( Qty ) in the second column to 600
    nlapiSetMatrixValue('price1', 'price', '2', 600);
    //Set values on line one. First you must select the line, then set all values,
    //then commit the line.
    nlapiSelectLineItem('price1', '1');
    nlapiSetCurrentLineItemMatrixValue('price1', 'price', 1, '11');
    nlapiSetCurrentLineItemMatrixValue('price1', 'price', 2, '12');
    nlapiCommitLineItem('price1');
}
```


[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetCurrentLineItemText(type, fldnam, text, firefieldchanged, synchronous)

Sets the value of a select field on the currently selected line using the display name. See also, [Using the Fire Field Changed Parameter](#).


Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set
- **text** {string} [required] - The display name associated with the value that the field is being set to
- **firefieldchanged** {boolean} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

 **Note:** The **firefieldchanged** parameter takes the values of **true** or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

 **Note:** In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void


[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)


Sets the value of the line-item field before it has been added to the form. Typically used in validate line functions. See also, [Using the Fire Field Changed Parameter](#).

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set
- **value** {string} [required] - The value the field is being set to.


 **Important:** Check box fields take the values of T or F, not **true** or **false**.

- **firefieldchanged** {boolean **true** || **false**} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**.

 **Note:** Available in Client SuiteScript only. See [Using the Fire Field Changed Parameter](#) for more information.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

 **Note:** In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetCurrentLineItemValues(type, fldnam, values, firefieldchanged, synchronous)

Sets the values for a multi-select sublist field. Note that like any other “set field” APIs, the values you use will be internal ID values. For example, rather than specifying 'Abe Simpson' as a customer value, you will use 232 or 88 or whatever the internal ID is for customer Abe Simpson.

However, if you are using this API to set the serialnumber field on the Item sublist, you will set the text string of the actual serial number, for example 'serialnum1', 'serialnum2', and so on.

This API is supported in client scripts only.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the multi-select sublist field being set.
- **values** {array} [required] - The values for the field.
- **firefieldchanged** {boolean} [optional] - If **true**, then the fieldchange script for that field is executed. If no value is provided, this argument defaults to **true**. (Available in Client SuiteScript only). See [Using the Fire Field Changed Parameter](#) for more information.

Note: The **firefieldchanged** parameter takes the values of *true* or **false**, not T or F.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.

Note: In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

Since

- Version 2012.1

Example

If the source of the items comes from different lot numbers, the best way of setting the serial number is the following. Note this is for client scripting only.

```
var serialArr = new Array();
serialArr[0] = 'amsLot1(1)';
serialArr[1] = 'amsLot2(1)';

nlapiSelectNewLineItem('item');
nlapiSetCurrentLineItemValue('item', 'item', 199, true, true);
nlapiSetCurrentLineItemValue('item', 'quantity', 2, true, true);
nlapiSetCurrentLineItemValues('item', 'serialnumbers', serialArr, true, true);
nlapiCommitLineItem('item');
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetLineItemValue(type, fldnam, linenum, value)

Sets the value of a sublist field on the current, **new** record. This API *can* be used in beforeLoad user event scripts to initialize sublist line items, but only on **new** records and only on non-stored sublist fields. If you execute this API on an existing record, nothing will happen.

Note that this API is supported in **user event scripts** only.

This function *can* be used in client SuiteScript, but note that **it is supported only on custom fields and the Description field**. If you use this function to set the value of a standard, built-in line item field, the function will not execute.

Note: Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **price** as the ID for the Pricing sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is 1 (not 0).
- **value** {string} [required] - The value the field is being set to

Returns

- void

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetMatrixValue(type, fldnam, column, value, firefieldchanged, synchronous)

This API is used to set a header field in a matrix sublist. This API is supported in client and user event scripts. It is typically used in pageInit (client) and beforeLoad (user event) events. Also note that this API should be used on matrix sublists only.

Note: Currently the Pricing sublist and the Demand Plan Detail sublist are the only matrix sublist types that support SuiteScript, and this API is used only for the Pricing sublist. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

In the case of the Pricing sublist, this API is used to set the quantity levels that appear in the Qty fields (see figure). Note that you should use this API only if you have the Quantity Pricing feature enabled in your account, as these header fields appear only if this feature is enabled. The following figure shows the header fields that can be set using nlapiSetMatrixValue:

PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 500	QTY 850	QTY 1,000	QTY
RRP						
B2B Online Price	-15.0%					
Corporate/Business/Government Customers	-7.5%					
Shopfront Rate	5.0%					
Online Price	-0.75%					

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The name of the field being set.
- **value** {string} [required] - The value the field is being set to .



Important: Check box fields take the values of T or F, not **true** or **false**.

- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).
- **firefieldchanged** {boolean **true** | **false**} [optional] - If **true**, then the field change script for that field is executed. If no value is provided, this argument defaults to **true**.



Note: Available in Client SuiteScript only. See [Using the Fire Field Changed Parameter](#) for more information.

- **synchronous** {boolean} [optional] - This parameter is relevant for client SuiteScripts only. In server scripts (such as user event scripts), this parameter will always execute as true.

In client scripts, if you do not set the value of **synchronous**, the default value is false, and the API executes asynchronously. If set to **true**, this API executes synchronously, which ensures a predictable script execution. Setting to **true** forces your client script to wait on any specified sourcing before continuing with the rest of the script.



Note: In client scripts, the **synchronous** parameter takes the values of **true** or **false**, not T or F.

Returns

- void

Since

- Version 2009.2

Example

The following sample is a user event script that executes on a beforeLoad event. This script is set to execute on the Pricing sublist on an Inventory Item record. On the Pricing sublist it will set the Base Price for the first two columns of the USA tab. The presence of the USA tab indicates that the Multiple Currencies feature is enabled in this account. Therefore, the internal ID of the type parameter in all matrix APIs will be price1.

```
function beforeLoad(type, form)
{
    nlapiSetFieldValue('itemid', '124');

    //Set the pricing matrix header field ( Qty ) in the second column to 600
    nlapiSetMatrixValue('price1', 'price', '2', 600);
    //Set values on line one. First you must select the line, then set all values,
    //then commit the line.
    nlapiSelectLineItem('price1', '1');
    nlapiSetCurrentLineItemMatrixValue('price1', 'price', 1, '11');
    nlapiSetCurrentLineItemMatrixValue('price1', 'price', 2, '12');
    nlapiCommitLineItem('price1');
```

```
}
```

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

nlobjSubList

See [nlobjSubList](#) - defined in the section on [UI Objects](#).

[Back to Sublist APIs](#) | [Back to SuiteScript Functions](#)

Using the Fire Field Changed Parameter

When creating scripts that provide the ability to watch a field for a change, and then write back to the field that changed, a risk of creating an infinite loop exists as follows:

1. The Client script watches for fieldA to change.
2. fieldA changes.
3. The script writes to fieldA, causing the Field Changed event to fire, returning the code to step 2, and this loop repeats indefinitely.

To prevent this looping behavior, you can set the optional **firefieldchanged** parameter in your client scripts.

The **firefieldchanged** parameter is available for all write functions. If set to **true**, the parameter causes any field changed events to fire as normal. This is the default setting. If set to **false**, field changed events are NOT fired.

Using the **firefieldchanged** parameter, you can modify the above example to:

1. Client script watches for fieldA to change.
2. fieldA changes.
3. Client script writes to fieldA using `firefieldchanged = false`, so the Field Changed event does not fire.

The following API calls can set the **firefieldchanged** parameter.

Note: The set line item text and value functions are NOT affected, as these do not currently call field changed after firing.

- [nlapiSetFieldValue\(fldnam, value, firefieldchanged, synchronous\)](#)
- [nlapiSetFieldText\(fldname, txt, firefieldchanged, synchronous\)](#)
- [nlapiSetCurrentLineItemValue\(type, fldnam, value, firefieldchanged, synchronous\)](#)
- [nlapiSetCurrentLineItemText\(type, fldnam, text, firefieldchanged, synchronous\)](#)

Note: The **firefieldchanged** parameter is provided for convenience. To prevent this loop, you could also include code that either checks to ensure that you are not writing the same value to the field or that tracks whether you wrote to the field.

Search APIs


For an overview of using SuiteScript to execute searches in NetSuite, see the help topic [SuiteScript 1.0 Searching Overview](#).

All APIs listed below are in alphabetical order.

- [nlapiCreateSearch\(type, filters, columns\)](#)
- [nlapiLoadSearch\(type, id\)](#)
- [nlapiLookupField\(type, id, fields, text\)](#)
- [nlapiSearchDuplicate\(type, fields, id\)](#)
- [nlapiSearchGlobal\(keywords\)](#)
- [nlapiSearchRecord\(type, id, filters, columns\)](#)
- [nlobjSearchColumn](#)
- [nlobjSearchFilter](#)
- [nlobjSearchResult](#)


nlapiCreateSearch(type, filters, columns)

Creates a new search. The search can be modified and run as an on demand search, without saving it. Alternatively, calling [nlobjSearch.saveSearch\(title, scriptId\)](#) will save the search to the database, so it can be reused later in the UI or using [nlapiLoadSearch\(type, id\)](#).

 **Note:** This function is agnostic in terms of its **filters** argument. It can accept input of either a search filter ([nlobjSearchFilter](#)), a search filter list ([nlobjSearchFilter\[\]](#)), or a search filter expression ([Object\[\]](#)).

Parameters

- **type** {string} [required] - The record internal ID of the record type you are searching (for example, customer | lead | prospect | partner | vendor | contact). For a list of internal IDs, in the NetSuite Help Center see the help topic [SuiteScript Supported Records](#).
- **filters** {[nlobjSearchFilter](#) | [nlobjSearchFilter\[\]](#) | [Object\[\]](#)} [optional] - A single [nlobjSearchFilter](#) object - **or** - an array of [nlobjSearchFilter](#) objects - **or** - a search filter expression.

 **Note:** You can further filter the returned [nlobjSearch](#) object by passing additional filter values. You will do this using the [nlobjSearch.addFilter\(filter\)](#) method or [nlobjSearch.addFilters\(filters\)](#) method.

- **columns** {[nlobjSearchColumn](#) or [nlobjSearchColumn\[\]](#)} [optional] - A single [nlobjSearchColumn\(name, join, summary\)](#) object - **or** - an array of [nlobjSearchColumn\(name, join, summary\)](#) objects. Note that you can further filter the returned [nlobjSearch](#) object by passing additional search return column values. You will do this using the [nlobjSearch.setColumns\(columns\)](#) method.

Returns

- [nlobjSearch](#)

Since

- Version 2012.1

Example 1

This example shows how to create a new saved search. First you define any search filters and search return columns. Next you call **nlapiCreateSearch** to execute the search. To save the search, you must then call the [nlobjSearch.saveSearch\(title, scriptId\)](#) method. Note that you are not required to save searches that are generated through **nlapiCreateSearch**.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
columns[3] = new nlobjSearchColumn( 'projectedamount' );
columns[4] = new nlobjSearchColumn( 'probability' );
columns[5] = new nlobjSearchColumn( 'email', 'customer' );
columns[6] = new nlobjSearchColumn( 'email', 'salesrep' );
// Create the saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch( 'My Opportunities in Last 90 Days', 'customsearch_kr' );
```

Example 2

This example shows how to load an existing search, create a new search based on existing criteria, define additional criteria, and then save the search as a new search.

```
var search = nlapiLoadSearch( 'opportunity', 'customsearch_blackfriday' );
var newSearch = nlapiCreateSearch( search.getSearchType(), search.getFilters(),
    search.getColumns() );
newSearch.addFilter( new nlobjSearchFilter( ... ) ); //Specify your own criteria here to add as a filter
newSearch.setIsPublic( true );
newSearch.saveSearch( 'My new opp search', 'customsearch_blacksaturday' );
```

Example 3

This example shows how to create a new saved search using a search filter expression.

```
//Define search filter expression
var filterExpression = [ [ 'trandate', 'onOrAfter', 'daysAgo90' ],
    'and',
    [ 'projectedamount', 'between', 1000, 100000 ],
    'and',
    [ 'customer.salesrep', 'anyOf', -5 ] ];

//Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
columns[3] = new nlobjSearchColumn( 'projectedamount' );
columns[4] = new nlobjSearchColumn( 'probability' );
columns[5] = new nlobjSearchColumn( 'email', 'customer' );
columns[6] = new nlobjSearchColumn( 'email', 'salesrep' );

//Create the saved search
var search = nlapiCreateSearch( 'opportunity', filterExpression, columns );
var searchId = search.saveSearch( 'My Opportunities in Last 90 Days', 'customsearch_kr' );
```

Example 4

This example shows how to load an existing search, create a new search based on existing criteria with the use of a search filter expression, define additional criteria, and then save the search as a new search.

```
var search = nlapiLoadSearch('opportunity', 'customsearch_blackfriday');
var newSearch = nlapiCreateSearch(search.getSearchType(), search.getFilterExpression(), search.getColumns());
newSearch.addFilter (new nlobjSearchFilter(.)); //Specify your own criteria here to add as a filter
newSearch.setIsPublic(true);
newSearch.saveSearch('My new opp search', 'customsearch_blacksaturday');
```

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

nlapiLoadSearch(type, id)

Loads an existing saved search. The saved search could have been created using the UI, or created using [nlapiCreateSearch\(type, filters, columns\)](#) in conjunction with [nlobjSearch.saveSearch\(title, scriptId\)](#).

Executing this API consumes 5 governance units.

Parameters

- **type** {string} [optional] - The record internal ID of the record type you are searching (for example, customer|lead|prospect|partner|vendor|contact). This parameter is case-insensitive. For a list of internal IDs, in the NetSuite Help Center see the help topic [SuiteScript Supported Records](#).
- **id** {string} [required] - The internal ID or script ID of the saved search. The script ID of the saved search is required, regardless of whether you specify the search **type**. If you do not specify the search **type**, you must set **type** to **null** and then set the script/search ID. See [Example 3](#) for more details.

Returns

- [nlobjSearch](#)

Since

- Version 12.1

Example 1

This sample shows how to load an existing saved search and add additional filtering criteria to the search. The search is then designated as a public search and saved.

```
var s = nlapiLoadSearch('opportunity', 'customsearch_blackfriday');
s.addFilter(new nlobjSearchFilter(...));
s.setIsPublic(true);
s.saveSearch('My new opp search', 'customsearch_blackfriday');
```

Example 2

This sample shows how to load an existing search, create a new search based on existing criteria, define additional criteria, and then save the search as a new search.

```
var search = nlapiLoadSearch('opportunity', 'customsearch_blackfriday');
var newSearch = nlapiCreateSearch(search.getSearchType(), search.getFilters(),
    search.getColumns());
newSearch.addFilter(new nlobjSearchFilter(...));
```



```
newSearch.setIsPublic(true);
newSearch.saveSearch('My new opp search', 'customsearch_blacksaturday');
```

Example 3

With the **type** parameter optional, developers have the flexibility to load existing searches, or execute new or existing searches without knowing the record type of the search.

A user can select a saved search from a custom saved search field. As a developer, you can have a user event script that loads or re-executes the selected search after the user saves the record. In this scenario, your script does not have access to the record type of the saved search. Your code has access only to the saved search ID, which is the value of My Saved Search Field. After you get the ID of the search, you can then pass in the ID to either **nlapiLoadSearch** or **nlapiSearchRecord**, depending on whether you want to load an existing search or re-execute it.

The following snippet shows how to get the ID of the saved search and then re-execute it, without having to specify the record type of the search.



Important: If you do not specify the search **type**, you must set **type** to null and then set the search ID.

```
var searchID = nlapiGetFieldValue('custentity_mysavedsearch');
var results = nlapiSearchRecord(null, searchID);
```

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

nlapiLookupField(type, id, fields, text)

See [nlapiLookupField\(type, id, fields, text\)](#) - also listed in the section [Field APIs](#).

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

nlapiSearchDuplicate(type, fields, id)

Performs a search for duplicate records based on the account's Duplicate Detection configuration. Note that this API only works for records that support duplicate record detection. These records include customers, leads, prospects, contacts, partners, and vendors.

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **type** {string} [required] - The record internal ID name you are checking duplicates for (for example, customer|lead|prospect|partner|vendor|contact). In the NetSuite Help Center, see the help topic [SuiteScript Supported Records](#).
- **fields** {string[]} [optional] - The internal ID names of the fields used to detect duplicate (for example, companyname|email|name|phone|address1|city|state|zipcode). Depending on the use case, **fields** may or may not be a required argument. If you are searching for duplicates based on the fields that appear on a certain record type, **fields** would be a **required** argument. If you are searching for the duplicate of a specific record (of a specified type), you would set **id** and not set **fields**.
- **id** {int} [optional] - internalId of existing record. Depending on the use case, **id** may or may not be a required argument. If you are searching for a specific record of a specified type, you must set **id**. If you are searching for duplicates based on field names, you will not set **id**; you will set **fields**.

Returns

- **{nlobjSearchResult[]}** - An Array of [nlobjSearchResult](#) objects corresponding to the duplicate records.



Important: Results are limited to 1000 records. Note that if there are no search results, **null** is returned.

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

Example

The following example performs a duplicate detection search for all customer records using the “email” field of the currently submitted record.

```
var fldMap = new Array();
fldMap['email'] = nlapiGetFieldValue('email');
var duplicateRecords = nlapiSearchDuplicate('customer', fldMap);
for (var i = 0; i < duplicateRecords.length; i++)
{
    var duplicateRecord = duplicateRecords[i];
    var record = duplicateRecord.getId();
    var rectype = duplicateRecord.getRecordType();
}
```

nlapiSearchGlobal(keywords)

Performs a global search against a single keyword or multiple keywords. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts. Usage metering allowed for **nlapiSearchGlobal** is 10 units.

Parameters

- **keywords** {string} [required] - Global search keywords string or expression

Returns

- An Array of [nlobjSearchResult](#) objects containing the following four columns: name, type (as shown in the UI), info1, and info2.



Important: Results are limited to 1000 rows. Note that if there are no search results, **null** is returned.

Example

The following example performs a global search for all records with the keyword **simpson**.

```
var searchresults = nlapiSearchGlobal( 'simpson' );
for ( var i = 0; i < searchresults.length; i++ )
{
    var searchresult = searchresults[ i ];
    var record = searchresult.getId( );
}
```

```

var rectype = searchresult.getRecordType( );

var name = searchresult.getValue( 'name' );
var type = searchresult.getValue( 'type' );
var info1 = searchresult.getValue( 'info1' );
var info2 = searchresult.getValue( 'info2' );
}

```

In the UI, the results returned from the snippet would look similar to the following:

Global Search: Results				
FILTERS				
EDIT VIEW	TYPE	NAME/ID	ADDITIONAL INFO 1	ADDITIONAL INFO 2
Edit View	Customer	Abe Simpson	Abe Simpson	504-231-1111(main) 555-1234(alt)
Edit View	Opportunity	OPP10059	Abe Simpson	Remodeling
Edit View	Opportunity	OPP10068	Abe Simpson	6/29/2006
Edit View	Opportunity	OPP10076	Abe Simpson	7/25/2006
Edit View	Opportunity	OPP10081	Abe Simpson	7/25/2006
Edit View	Opportunity	OPP10082	Abe Simpson	7/25/2006
Edit View	Opportunity	OPP10086	Abe Simpson	7/26/2006
Edit View	Opportunity	OPP10089	Abe Simpson	1/2/2007
Edit View	Opportunity	OPP10109	Abe Simpson	8/18/2008
Edit View	Project	install new tires (Abe Simpson)		
Edit View	Project	Installation (SORD10083) (Abe Simpson)		
Edit View	Project	Installation (SORD10083)jobs test 2) (Abe Simpson)		
Edit View	Project	Installation (SORD10094)jobs test 1) (Abe Simpson)		



Note: This screenshot displays the NetSuite user interface that was available before Version 2010 Release 2.

Note that as with global search functionality in the UI, you can programmatically filter the global search results that are returned. In the snippet above, if your first line of code looked like this:

```
var searchresults = nlapiSearchGlobal( ' cu: simpson' );
```

only the three Abe Simpson customer records will be returned in your search. For more general information about global search in NetSuite, see the help topic [Global Search](#) in the NetSuite Help Center.

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

nlapiSearchRecord(type, id, filters, columns)

Performs a search using a set of criteria (your search filters) and columns (the results). Alternatively, you can use this API to execute an existing saved search.



Important: When you use this API, keep the following considerations in mind:

- Results are limited to 1000 rows.
- In search/lookup operations, custom fields of type "long text" are truncated. In accounts with multiple languages enabled, the returned value is truncated at 1,300 characters. In accounts that don't use multiple languages, the field return truncates at 3,900 characters.

Usage metering allowed for `nlapiSearchRecord` is 10 units.

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Note: This API can also be used to search custom lists. In the NetSuite Help Center, see the help topic [Searching Custom Lists](#) for an example.

You can extract the desired information from the search results using the methods available on the returned `nlobjSearchResult` object.

Note that results returned by `nlapiSearchRecord` are **not sortable** directly. However, you can accomplish sorting using either of the following methods:

1. Use the `setSort` function on an `nlobjSearchColumn` object(s), and then pass the sorted `nlobjSearchColumn` object(s) to the `columns` parameter. See [setSort\(order\)](#) for an example.
2. Reference a saved search that is sorted by `internalid` or `internalidnumber`
3. Sort the array of results that is returned in JavaScript using a custom Array sorting function. See the topic called "Creating, displaying, and sorting an array" at <http://developer.mozilla.org/>.

Note: This function is agnostic in terms of its `filters` argument. It can accept input of either a search filter (`nlobjSearchFilter`), a search filter list (`nlobjSearchFilter[]`), or a search filter expression (`Object[]`).

Parameters

- **type** {string} [optional] - The record internal ID of the record type you are searching. For a list of internal IDs, in the NetSuite Help Center see the help topic [SuiteScript Supported Records](#).
- **id** {int | string} [optional] - The `internalId` or custom `scriptId` for the saved search. To obtain the `internalId`, go to Lists > Search > Saved Searches. The `internalId` appears in the Internal ID column. If you have created a custom `scriptId` when building your search, this ID will appear in the ID column.

Note the following about how this argument is validated:

- If the `internalId` or `scriptId` is valid, the saved search is executed (assuming the search has no user or role restrictions applied to it).
- If you do not specify the search **type**, the `id` parameter **becomes REQUIRED**. In this case, you must set **type** to **null** and then specify the `scriptId` for the saved search. See [Example 3](#) for an example of when and type you might create this type of script.
- If there is no `internalId` or `scriptId` (null or empty string or left out altogether), an on demand search will be executed and this argument will be ignored.
- If the `internalId` or `scriptId` is invalid, the following user error is thrown: That search or mass updates does not exist.
- **filters** {`nlobjSearchFilter` | `nlobjSearchFilter[]` | `Object[]`} [optional] - A single `nlobjSearchFilter` object - **or** - an array of `nlobjSearchFilter` objects - **or** - a search filter expression.

Note: You can further filter the returned saved search by passing additional filter values.

- **columns** {`nlobjSearchColumn` or `nlobjSearchColumn[]`} [optional] - A single `nlobjSearchColumn(name, join, summary)` object - **or** - an array of `nlobjSearchColumn(name, join, summary)` objects. Note that you can further filter the returned saved search by passing additional search return column values.

Returns

- {`nlobjSearchResult[]`} - An array of `nlobjSearchResult` objects corresponding to the searched records.



Important: The array returned by this API is **read-only**. Note that if there are no search results, **null** is returned.

Throws

- SSS_INVALID_RECORD_TYPE
- SSS_TYPE_ARG_REQD
- SSS_INVALID_SRCH_ID
- SSS_INVALID_SRCH_FILTER
- SSS_INVALID_SRCH_FILTER_JOIN
- SSS_INVALID_SRCH_OPERATOR
- SSS_INVALID_SRCH_COL_NAME
- SSS_INVALID_SRCH_COL_JOIN
- SSS_INVALID_SRCH_FILTER_EXPR
- SSS_INVALID_SRCH_FILTER_EXPR_DANGLING_OP
- SSS_INVALID_SRCH_FILTER_EXPR_OBJ_TYPE
- SSS_INVALID_SRCH_FILTER_EXPR_PAREN_DEPTH
- SSS_INVALID_SRCH_FILTER_LIST_PARENS
- SSS_INVALID_SRCH_FILTER_LIST_TERM

Examples

For **code samples** showing the kinds of searches you can execute using the `nlapiSearchRecord` function, see the help topic [Search Samples](#) in the NetSuite Help Center. If you are new to searching with SuiteScript, also see the help topic [SuiteScript 1.0 Searching Overview](#).

Back to [Search APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjSearchColumn

See `nlobjSearchColumn(name, join, summary)` - defined in the section on [Standard Objects](#).

Back to [Search APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjSearchFilter

See `nlobjSearchFilter` - defined in the section on [Standard Objects](#).

Back to [Search APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjSearchResult

See `nlobjSearchResult` - defined in the section on [Standard Objects](#).

[Back to Search APIs](#) | [Back to SuiteScript Functions](#)

Scheduling APIs

The scheduling APIs are used to start, gather information about, and pause scripts until a more appropriate time.

- `nlapiScheduleScript(scriptId, deployId, params)`
- `nlapiSetRecoveryPoint()`
- `nlapiYieldScript()`

For a complete overview of working with scheduled scripts in NetSuite, see the help topic [Scheduled Scripts](#).

`nlapiScheduleScript(scriptId, deployId, params)`

A call to this API submits a scheduled script for processing. For this to work, the scheduled script must have a status of **Not Scheduled** on the Script Deployment page. If the script's status is set to **Testing** on the Script Deployment page, the API does not submit the script for processing.

If the deployment status on the Script Deployment page is set to **Scheduled**, the script is submitted for processing according to the time(s) specified on the Script Deployment page.

The `nlapiScheduleScript` API consumes 20 units per call. This API is supported in user event, portlet, RESTlet, scheduled, and Suitelet scripts.



Important: There is **no** unit metering if you resubmit the current script instance (see [Example 1 - Rescheduling a Script](#)). Note, however, `nlapiScheduleScript` is still 20 units per call if you are trying to submit **other** scripts for processing.

One or more calls to `nlapiScheduleScript` can be made from Suitelet, RESTlet, user event, and portlet scripts. For example, to submit a scheduled script for processing from another script type, such as a user event script or a Suitelet.

Note that you can **also** call `nlapiScheduleScript` from within a **scheduled script** to:

1. Create another instance of the currently-executing scheduled script, and submit it for processing.
2. Submit another scheduled script for processing.



Note: Only administrators can run scheduled scripts. If a user event script calls `nlapiScheduleScript`, the user event script has to be deployed with admin permissions.

For additional details specific to using `nlapiScheduleScript`, see the help topic [Using nlapiScheduleScript to Submit a Script for Processing](#).

For general details on working with scheduled scripts, see the help topic [Overview of Scheduled Script Topics](#).

Parameters

- `scriptId` {string | int} [required] - The script internalId or custom scriptId

- **deployId** {string | int} [optional] - The deployment internal ID or script ID. The first “free” deployment will be used. Free means that the script's deployment status appears as Not Scheduled and the deployment is not currently executing.



Important: The **deployId** is a **required argument** if you are calling **nlapiScheduleScript** to resubmit a scheduled script that is currently executing.

- **params** {Object} [optional] - Object of name/values used in this scheduled script instance - used to override the script parameters values for this execution.

Note that name values are the script parameter internal IDs. If you are not familiar with what a script parameter is in the context of SuiteScript, see the help topic [Creating Script Parameters Overview](#) in the NetSuite Help Center.

Returns

- A string whose value is QUEUED if the script was successfully submitted for processing by this call, or it returns the script's current status. Valid status values are:
 - **QUEUED** - The script you requested is already submitted for processing and waiting to be run. This script cannot be requested again until it finishes processing. If the script is INQUEUE, you must try again later if you want to run the script.
 - **INPROGRESS** - The scheduled script is currently running.
 - **SCHEDULED** - The script's deployment status is set to scheduled and will be submitted for processing according to the time(s) specified on the script deployment.



Important: This API returns NULL if the scheduled script is undeployed or invalid.

Example 1 - Rescheduling a Script

Use **nlapiScheduleScript**, **nlobjContext.getScriptId** and **nlobjContext.getDeploymentId** to resubmit the currently executing scheduled script if there are more sales orders to update when the unit usage limit is reached.



Important: There is **no** unit metering if you are resubmitting the current script instance. In the following sample, **nlapiScheduleScript** consumes no units. Note, however, that **nlapiScheduleScript** is still 20 units per call if you are submitting **other** scripts.

```
function updateSalesOrders()
{
    var context = nlapiGetContext();
    var searchresults = nlapiSearchRecord('salesorder', 'customscript_orders_to_update')
    if ( searchresults == null )
        return;
    for ( var i = 0; i < searchresults.length; i++ )
    {
        nlapiSubmitField('salesorder', searchresults[i].getId(), 'custbody_approved', 'T')
        if ( context.getRemainingUsage() <= 0 && (i+1) < searchresults.length )
        {
            var status = nlapiScheduleScript(context.getScriptId(), context.getDeploymentId())
            if ( status == 'QUEUED' )
                break;
        }
    }
}
```

```
}
```

Example 2

See more examples in the section [Scheduled Script Samples](#).

[Back to Scheduling APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetRecoveryPoint()

Creates a recovery point saving the state of the script's execution. When NetSuite resumes execution of the script, it resumes the script at the specified recovery point. Also note that when the script is resumed, its governance units are reset. Be aware, however, all scheduled scripts have a 50 MB memory limit. For complete details on scheduled script memory limits, see [Understanding Memory Usage in Scheduled Scripts](#).

A typical implementation for this API might be as follows. Based on the status returned by **nlapiSetRecoveryPoint**, the script executes different logic.

```
res = nlapiSetRecoveryPoint()
if (res.status == 'FAILURE')
    examine the reason and either cleanup/try again OR exit
else if (res.status == 'SUCCESS')
    do X
else if (res.status == 'RESUME')
    examine the reason and react appropriately
do Z
do A
```

Note that you can use **nlapiSetRecoveryPoint** in conjunction with **nlapiYieldScript** to effectively pause the script until a later time when it is more appropriate to run the script.



Important: This API can only be called from scheduled scripts. Calling this API from any other script type results in an error.



Important: Scripts that contain live references to a record must null those references before they call **nlapiYieldScript** or **nlapiSetRecoveryPoint**. If the references are not nulled and the record was deleted or transformed, the scripts return an SSS_SCRIPT_DESERIALIZATION_FAILURE error.



Important: Scripts that contain live references to files larger than 5MB must null the references before they call **nlapiYieldScript** or **nlapiSetRecoveryPoint**. If these references are not nulled, the script returns an SSS_FILE_OBJECT_NOT_SERIALIZABLE error. See [Example – Nulling a reference to a file larger than 5MB](#) for an example.



Important: This API is not supported within JavaScript array iteration functions (such as **every**, **filter**, **forEach**, **map** and **some**). JavaScript array iteration functions are designed to be executed as a whole. SuiteScript cannot yield in the middle of these control structures. To work around this limitation, use a **for** loop or the **forEachResult(callback)** function. For more information about **nlobjSearchResultSet.forEachResult(callback)**, see [forEachResult\(callback\)](#).

The **nlapiSetRecoveryPoint** API consumes 100 units per call.

For an overview of possible use cases for setting recovery points in your scheduled scripts, see [Setting Recovery Points in Scheduled Scripts](#).

Returns

- Native Javascript Object
 - status {string}
 - SUCCESS – Save point was created.
 - FAILURE – The recovery point was unable to be created. Returns the reason for the failure and the footprint size of the script.
 - RESUME – Script is being resumed.
 - reason {string}
 - SS_NLAPIYIELDSCRIPT - Yield was called.
 - SS_ABORT -The JVM unintentionally stopped (native error, no response, etc.) --mimics normal "ABORT" states.
 - SS_MAJOR_RELEASE – A major NetSuite release is pending, processes are being stopped.
 - SS_EXCESSIVE_MEMORY_FOOTPRINT – The saved object is too big.
 - SS_CANCELLED – A user requested that the script stop.
 - SS_DISALLOWED_OBJECT_REFERENCE – The script is attempting to serialize an object that is not serializable (see Supported Objects).
 - SSS_FILE_OBJECT_NOT_SERIALIZABLE – The script is attempting to serialize an nlobjFile object that references a file larger than 5MB.
 - SSS_SCRIPT_DESERIALIZATION_FAILURE – Scheduled Script deserialization failure, Failed to recover record state: There are no records of this type. There was a problem deserializing record 119 (serviceitem). Probable cause was its deletion or transformation. To prevent this, please assign null to variable holding this nlobjRecord reference before yielding in scheduled script.
 - size {integer} – The size of the saved object.
 - information {string} – Additional information about the status.




Important: If `nlapiYieldScript` or `nlapiSetRecoveryPoint` returns a FAILURE with `SS_DISALLOWED_OBJECT_REFERENCE`, the object type will be stored in the information property. To fix this problem, find the offending reference and set it to null.

Supported Objects:

All JavaScript native types, plus:


- `nlobjConfiguration`
- `nlobjContext`
- `nlobjError`
- `nlobjFile` (files up to 5BM in size)
- `nlobjRecord`
- `nlobjSubrecord`
- `nlobjSearchColumn`
- `nlobjSearchFilter`
- `nlobjSearchResult`

- `nlobjSearchResultCell`
- all 3rd party XML Library objects

 **Important:** All other object types are not supported.

Example – Setting a recovery point, handling errors, and resuming a script

The following sample shows a scheduled script that runs a customer search. The script iterates through the results of the customer search, and after every five records, sets a recovery point. If there is an unexpected server failure, the script will resume from the current "i" index of the search results.

 **Note:** The `handleCustomer` function in this script is not defined. The function is there only to demonstrate generic processing you could do with search results.

This script also checks the governance of the script. If the script goes above the governance threshold, the script is yielded. Based on the status returned by `setRecoveryPoint`, an execution log is created to document the reason this script was resumed. And based on the reason, a more descriptive text message is thrown to the user. Note that if the reason is `SS_EXCESSIVE_MEMORY_FOOTPRINT` the `cleanUpMemory` function is executed and an additional recovery point is set.

```
function runScheduledScript(status, queueid)
{
    var records = nlapiSearchRecord('customer', 15);

    for( var i = 0; i < records.length; i++ )
    {
        handleCustomer(records[i].getRecordType(), records[i].getId());

        if( (i % 5) == 0 ) setRecoveryPoint(); //every 5 customers, we want to set a recovery point so that, in case
        of an unexpected server failure, we resume from the current "i" index instead of 0

        checkGovernance();
    }
}

function setRecoveryPoint()
{
    var state = nlapiSetRecoveryPoint(); //100 point governance
    if( state.status == 'SUCCESS' ) return; //we successfully create a new recovery point
    if( state.status == 'RESUME' ) //a recovery point was previously set, we are resuming due to some unforeseen
    error
    {
        nlapiLogExecution("ERROR", "Resuming script because of " + state.reason + ". Size = " + state.size);
        handleScriptRecovery();
    }
    else if ( state.status == 'FAILURE' ) //we failed to create a new recovery point
    {
        nlapiLogExecution("ERROR", "Failed to create recovery point. Reason = " + state.reason + " / Size = " +
        state.size);
        handleRecoveryFailure(state);
    }
}

function checkGovernance()
```

```

{
    var context = nlapiGetContext();
    if( context.getRemainingUsage() < myGovernanceThreshold )
    {
        var state = nlapiYieldScript();
        if( state.status == 'FAILURE' )
        {
            nlapiLogExecution("ERROR","Failed to yield script, exiting: Reason = "+state.reason + " / Size = "+
state.size);
            throw "Failed to yield script";
        }
        else if ( state.status == 'RESUME' )
        {
            nlapiLogExecution("AUDIT", "Resuming script because of " + state.reason+" Size = "+ state.size);
        }
        // state.status will never be SUCCESS because a success would imply a yield has occurred. The equivalent
        response would be yield
    }
}

function handleRecoverFailure(failure)
{
    if( failure.reason == 'SS_MAJOR_RELEASE' ) throw "Major Update of NetSuite in progress, shutting down all
processes";
    if( failure.reason == 'SS_CANCELLED' ) throw "Script Cancelled due to UI interaction";
    if( failure.reason == 'SS_EXCESSIVE_MEMORY_FOOTPRINT' ) { cleanUpMemory(); setRecoveryPoint(); } //avoid infinite
loop
    if( failure.reason == 'SS_DISALLOWED_OBJECT_REFERENCE' ) throw "Could not set recovery point because of a
reference to a non-recoverable object: "+ failure.information;
}

function cleanUpMemory(){...set references to null, dump values seen in maps, etc}

```

Example – Nulling a reference to a file larger than 5MB

The following example assumes that the `nlobjFile` object, `largeFile`, references a file larger than 5MB. Scripts that contain live references to files larger than 5MB must null the references before they call `nlapiYieldScript` or `nlapiSetRecoveryPoint`. If these references are not nulled, the script returns an `SSS_FILE_OBJECT_NOT_SERIALIZABLE` error.

```

var largeFile = nlapiLoadRecord('1234');
var pdf = nlapiXMLToPDF(largeFile);
largeFile = null;
nlapiYieldScript();
//perform additional logic after points refreshed

```

Back to [Scheduling APIs](#) | Back to [SuiteScript Functions](#)

nlapiYieldScript()

Creates a recovery point and, when executed, continues from the recovery point. The newly resubmitted script has its governance units reset. To summarize, `nlapiYieldScript` works as follows:

1. Creates a new recovery point.

2. Creates a new scheduled script instance with governance reset.
3. Associates the recovery point to the new instance of the scheduled script
4. Submits the new instance for processing.

Note: If the original script instance uses a queue, the new instance will use the same queue. If the original script instance does not use a queue, the new instance will be processed by the next available processor according to priority and submission time.

Note: If the yield call fails, a FAILURE status will be returned. On success, the call does not return until the script is resumed.

Calling this function consumes no governance units. Note also, calling this API resets the unit counter for the currently executing script. Be aware, however, all scheduled scripts have a 50 MB memory limit. Calling this API will not reset the memory size of the script to 0. It only resets the governance units. For complete details on scheduled script memory limits, see [Understanding Memory Usage in Scheduled Scripts](#).

Important: This API can only be called from scheduled scripts. Calling this API from any other script type will result in an error.

Important: Scripts that contain live references to a record must null those references before they call `nlapiYieldScript` or `nlapiSetRecoveryPoint`. If the references are not nulled and the record was deleted or transformed, the scripts return an `SSS_SCRIPT_DESERIALIZATION_FAILURE` error.

Important: Scripts that contain live references to files larger than 5MB must null the references before they call `nlapiYieldScript` or `nlapiSetRecoveryPoint`. If these references are not nulled, the script returns an `SSS_FILE_OBJECT_NOT_SERIALIZABLE` error. See [Example – Nulling a Reference to a File Larger than 5MB](#) for an example.

Important: This API is not supported within JavaScript array iteration functions (such as `every`, `filter`, `forEach`, `map` and `some`). JavaScript array iteration functions are designed to be executed as a whole. SuiteScript cannot yield in the middle of these control structures. To work around this limitation, use a `for` loop or the `forEachResult(callback)` function. For more information about `nlobjSearchResultSet.forEachResult(callback)`, see [forEachResult\(callback\)](#).

Returns

- Native Javascript Object
 - status {string}
 - FAILURE – The recovery point was unable to be created. Returns the reason for the failure and the footprint size of the script.
 - RESUME – Script is being resumed.
 - reason {string}
 - SS_NLAPIYIELDSCRIPT - Yield was called.
 - SS_ABORT -The JVM unintentionally stopped (native error, no response, etc.) --mimics normal "ABORT" states.
 - SS_MAJOR_RELEASE – A major NetSuite release is pending, processes are being stopped.
 - SS_EXCESSIVE_MEMORY_FOOTPRINT – The saved object is too big.

- `SS_CANCELLED` – A user requested that the script stop.
- `SS_DISALLOWED_OBJECT_REFERENCE` – The script is attempting to serialize an object that is not serializable (see Supported Objects).
- `SSS_FILE_OBJECT_NOT_SERIALIZABLE` – The script is attempting to serialize an `nlobjFile` object that references a file larger than 5MB.
- `SSS_SCRIPT_DESERIALIZATION_FAILURE` – Scheduled Script deserialization failure, Failed to recover record state: There are no records of this type. There was a problem deserializing record 119 (serviceitem). Probable cause was its deletion or transformation. To prevent this, please assign null to variable holding this `nlobjRecord` reference before yielding in scheduled script.
- `size {integer}` – The size of the saved object.
- `information {string}` – Additional information about the status.
- Be careful if using this API within try / catch / finally. On a successful yield, all the finally blocks will be called, but catches will be ignored.
- It is advisable to use the final block for code which is not going to affect program flow, for example - writing log entries.
- If you have a yield in the try block, it is possible that some instructions in the finally block will execute before the yield takes place. The same instructions will execute again on resume.

Supported Objects:

All JavaScript native types, plus:

- `nlobjConfiguration`
- `nlobjContext`
- `nlobjError`
- `nlobjFile` (files up to 5MB in size)
- `nlobjRecord`
- `nlobjSubrecord`
- `nlobjSearchColumn`
- `nlobjSearchFilter`
- `nlobjSearchResult`
- `nlobjSearchResultCell`
- all 3rd party XML Library objects



Important: All other object types are not supported.

Example – Nulling a Reference to a File Larger than 5MB

The following example assumes that the `nlobjFile` object, `largeFile`, references a file larger than 5MB. Scripts that contain live references to files larger than 5MB must null the references before they call `nlapiYieldScript` or `nlapiSetRecoveryPoint`. If these references are not nulled, the script returns an `SSS_FILE_OBJECT_NOT_SERIALIZABLE` error.

```
var largeFile = nlapiLoadRecord('1234');
var pdf = nlapiXMLToPDF(largeFile);
largeFile = null;
```

```
nlapYieldScript();
//perform additional logic after points refreshed
```

[Back to Scheduling APIs](#) | [Back to SuiteScript Functions](#)

Execution Context APIs

Context APIs are used to get system information or metadata about a script that is running, a user in a NetSuite account, or certain settings that have been applied to account.

All APIs listed below are in alphabetical order.

- [nlapiGetContext\(\)](#)
- [nlapiGetDepartment\(\)](#)
- [nlapiGetLocation\(\)](#)
- [nlapiGetRole\(\)](#)
- [nlapiGetSubsidiary\(\)](#)
- [nlapiGetUser\(\)](#)
- [nlapiLogExecution\(type, title, details\)](#)
- [nlobjContext](#)

nlapiGetContext()

Used to branch scripts depending on the metadata or context of the execution. For example, you may want the script to perform in one way when a form is accessed via the UI and another when the form is accessed via SOAP web services.

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- [nlobjContext](#) object containing information (metadata) about the current user or script context. You must use the [nlobjContext.getSetting](#) method on [nlapiGetContext](#) to reference script parameters. For example, to obtain the value of a script parameter called `custscript_case_field`, you must use the following code:

```
nlapiGetContext().getSetting('SCRIPT', 'custscript_case_field')
```

Specifying SOAP Web Services Context

To cause a form to behave differently in SOAP web services versus the UI, you can do one of the following:

- Write context-specific SuiteScript code and use the [nlapiGetContext](#) function to branch the code
- Disable SuiteScript in web services

However, both Client and Server SuiteScripts are written to enforce customized business rules that may need to be enforced regardless of the mechanism by which a record is created or updated within NetSuite. This is particularly true for customers who deploy a SuiteCloud partner application and want to be sure their business rules are still respected. Since Client SuiteScript often has browser-specific behavior that requires user action and cannot automatically run during a SOAP web services

call, NetSuite recommends that you disable Client SuiteScript and deploy Server SuiteScript for those business conditions that need to be enforced in all cases.

To specify that Server SuiteScript should never execute during a SOAP web services call, enable the **Disable Server-side Scripting** preference on the web services Preference page at Setup > Integration > SOAP Web Services Preferences.



Important: Only enable this preference when data submitted via web services does NOT need to adhere to custom business logic and workflows that may be executed via Server SuiteScript.

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetDepartment()

This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- The integer value of the current user's department (for example, 3, 9, or 1)

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetLocation()

Returns the integer value of the current user's location. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- The integer value of the current user's location (for example, 5, 7, -2). Note that if a location has not been set, the value of **-1** is returned.

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetRole()

Returns the internalId for the current user's role. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- The integer value of the current user's role (for example: 1, 3, or 5). Note that the value of **-31** is returned if a user cannot be properly identified by NetSuite. This occurs when the user has not authenticated to NetSuite, for example when using externally available (**Available without Login**) Suitelets or online forms.

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetSubsidiary()

Returns the internalId for the current user's subsidiary. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- The integer value for the current user's subsidiary (for example 1, 3, or 5). Note that if a subsidiary has not been set (for example, the subsidiaries feature is not turned on in the user's account), the value of 1 is returned if this function is called.

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlapiGetUser()

Returns the internalId of the current NetSuite user. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Returns

- The integer value of the current user (for example, 195, 25, 21). Note that the value of **-4** is returned if a user cannot be properly identified by NetSuite. This occurs when the user has not authenticated to NetSuite, for example when using externally available (**Available without Login**) Suitelets or online forms.

Example

The following sample shows how to use **nlapiGetUser** in conjunction with **nlapiSendEmail**. In this sample, the internal ID of the currently logged in user is passed to the **author** argument in **nlapiSendEmail**, which is a required argument in this API.

```
function afterSubmitEmail(type)
{
    //User event script deployed to purchase orders.
    //Set the afterSubmit type to approve. As soon as the PO is
    //approved, an email is sent.
    if (type == 'approve')

    //Get the user ID of the person approving the PO. This will be the email author.
    var userId = nlapiGetUser();

    //Send an email to the supervisor, K. Wolfe in this case.
    var sendEmail = nlapiSendEmail(userId, 'kwolfe@netsuite.com', 'Purchase Order Notification', 'Purchase
    order approved', null, null, 'transaction', null);
}
```

See also


[nlapiSendEmail\(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo\)](#)

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)


nlapiLogExecution(type, title, details)

This API is supported in all server-side and record-level (global) client scripts.


Use this API to log an entry on the Execution Log subtab. The Execution Log subtab appears on the Script Deployment page for a script. See the help topic [Creating Script Execution Logs](#) to learn more about writing logs to the Execution Log subtab.

 **Note:** When you are debugging a script in the SuiteScript Debugger, log details appear on the Execution Log tab **of the SuiteScript Debugger**, NOT the script's Script Deployment page.


The log **type** argument is used in conjunction 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 **nlapiLogExecution** 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. See the help topic [Setting Script Execution Log Levels](#) for more information using the Log Level field.

 **Important:** Be aware that NetSuite governs the amount of logging that can be done by a company in any 60 minute time period. For complete details, see the help topic [Governance on Script Logging](#).

Also note that if the script's deployment status is set to Released, then the default Log Level is ERROR. If the status is set to Testing, the default Log Level is DEBUG.

 **Note:** The Execution Log tab also lists notes returned by NetSuite such as error messages. For additional information on using the Execution Log, see the help topic [Creating Script Execution Logs](#) in the NetSuite Help Center.

Parameters

 **Important:** The Script Deployment Execution Log does not support JavaScript execution or markup rendering. When passed to `nlapiLogExecution()`, JavaScript and markup (html, xml, etc.) appear as plain text on the Execution Log.

- **type** {string} [required] - One of the following log types:
 - DEBUG
 - AUDIT
 - ERROR
 - EMERGENCY
- **title** {string} [optional] - A title used to organize log entries (max length: 99 characters). If you set **title** to **null** or empty string ("), you will see the word "Untitled" appear in your log entry.
- **details** {string} [optional] - The details of the log entry (max length: 3999 characters)

Throws

- SSS_MISSING_REQD_ARGUMENT - if no value is specified for **title**.

Returns

- void

Example 1

This sample creates a new Customer record. When this script runs, execution details are logged on the Execution Log subtab on the Script Deployment page.

```
//Create a new Customer record
var newCust = nlapiCreateRecord('customer');

//Set the title field on the Customer record
```

```
newCust.setFieldValue('title', 'My New Customer');

var custId = nlapiSubmitRecord(newCust, true);
nlapiLogExecution('DEBUG', 'customer record created successfully', 'ID = ' + custId);
```

Example 2

This snippet shows a search against sales orders, based on specified search filters and search columns. After the search is complete, the remaining units for the script will be logged on the Execution Log tab. If you are worried that your script will exceed unit governance limits, it is useful to track unit usage in the Execution Log.

```
//Search for the sales orders with trandate of today
var todaySO = nlapiSearchRecord('salesorder', null, todaySOFilters, todaySOColumns);

nlapiLogExecution('DEBUG', 'Remaining usage after searching sales orders from today',
context.getRemainingUsage());
```

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

nlobjContext

See [nlobjContext](#) - defined in the section on [Standard Objects](#).

[Back to Execution Context APIs](#) | [Back to SuiteScript Functions](#)

UI Builder APIs

UI builder APIs allow developers to programmatically create various components of a the NetSuite UI (for example, forms, fields, sublists, tabs, portlets). You can also use the UI builder APIs to create NetSuite-looking assistant wizards.

For more details on working with UI builder APIs, see also [UI Objects Overview](#).

All APIs listed below are in alphabetical order.

- [nlapiCreateAssistant\(title, hideHeader\)](#)
- [nlapiCreateForm\(title, hideNavbar\)](#)
- [nlapiCreateList\(title, hideNavbar\)](#)
- [nlapiCreateTemplateRenderer\(\)](#)
- [nlobjAssistant](#)
- [nlobjAssistantStep](#)
- [nlobjButton](#)
- [nlobjColumn](#)
- [nlobjField](#)
- [nlobjFieldGroup](#)
- [nlobjForm](#)
- [nlobjList](#)
- [nlobjPortlet](#)
- [nlobjSubList](#)

- [nlobjTab](#)
- [nlobjTemplateRenderer](#)

nlapiCreateAssistant(title, hideHeader)

Use this function to return a reference to an [nlobjAssistant](#) object, which is the basis for building your own custom assistant. This API is supported in Suitelets.

Parameters

- **title** {string} [required] - The name of the assistant. This name will appear at the top of all assistant pages.
- **hideHeader** {boolean} [optional] - If not set, defaults to false. If set to true, the header (navbar/logo) on the assistant is hidden from view. Note that the header is where the Add to Shortcuts link appears.

Returns

- [nlobjAssistant](#) object

Since

- Version 2009.2

Example

This snippet shows how to call **nlapiCreateAssistant** to return a reference to the **nlobjAssistant** object. With the **nlobjAssistant** object instantiated, you can then define the steps of the assistant.

```
var assistant = nlapiCreateAssistant("Small Business Setup Assistant");
assistant.setOrdered(true); // indicate that all steps must be completed sequentially

assistant.addStep('companyinformation', 'Setup Company Information').setHelpText("Setup your
    <b>important</b> company information in the fields below.")

assistant.addFieldGroup('companyinfogroup', 'Company Information');
assistant.addField('companyname', 'text', 'Company Name', null, 'companyinfogroup');
assistant.addField('legalname', 'text', 'Legal Name', null, 'companyinfogroup');

assistant.addStep('entercontacts', 'Enter Contacts').setHelpText("Manually add contacts into your account.")
assistant.addStep('importdata', 'Import Data').setHelpText("Finally, import records into your account via
    CSV.");

response.writePage(assistant);
```

Small Business Setup Assistant

STEPS

- 1 Setup Company Information
- 2 Enter Contacts
- 3 Import Data

Setup Company Information

Setup your **important** company information in the fields below.

▼ **Company Information**

COMPANY NAME

LEGAL NAME

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateForm(title, hideNavbar)

Creates an [nlobjForm](#) object which can be used to generate an entry form page. This API is available to Suitelets only.

Parameters

- **title** {string} [required] - The title for the form
- **hideNavbar** {boolean} [optional] - Set to **true** if the navigation bar should be hidden on the Suitelet. Setting to **true** enables “popup page” use cases in which the popup can be created with the [UI Objects](#) API rather than HTML.
When **hideNavbar** is set to **false**, the standard NetSuite navigation appears on the form or popup. Note that this navigation bar contains links to pages that require users to be logged in to access.

Returns

- An [nlobjForm](#) object

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateList(title, hideNavbar)

Creates an [nlobjList](#) object used to generate an internal standalone list. This API is available to **Suitelets only**.

Parameters

- **title** {string} [required] - The title for the list
- **hideNavbar** {boolean} [optional] - Set to **true** if the navigation bar should be hidden on the Suitelet. Setting to **true** enables “popup page” use cases in which the popup can be created with the [UI Objects](#) API rather than HTML.
When **hideNavbar** is set to **false**, the standard NetSuite navigation appears on the form or popup. Note that this navigation bar contains links to pages that require users to be logged in to access.

Returns

- An [nlobjList](#) object

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlapiCreateTemplateRenderer()

Use this function to produce HTML and PDF printed forms that utilize advanced PDF/HTML template capabilities. This API returns an [nlobjTemplateRenderer](#) object. 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 an [nlobjResponse](#) object, or as XML string that can be passed to [nlapiXMLToPDF\(xmlstring\)](#) to produce a PDF.

This function is available when the Advanced PDF/HTML Templates feature is enabled. For information about this feature, see the help topic [Advanced PDF/HTML Templates](#).

Note: The advanced template API expects your template string to conform to FreeMarker syntax. Refer to <http://freemarker.sourceforge.net/docs/index.xml> for details.

Returns

- An `nlobjTemplateRenderer` object

Since

- Version 2013.1

Example

```
function renderRecord(request, response)
{
    var salesOrderID = 3;
    var salesOrder = nlapiLoadRecord('salesorder', salesOrderID);
    var renderer = nlapiCreateTemplateRenderer();
    renderer.setTemplate(.);
    renderer.addRecord('record', salesOrder);
    response.setContentType('HTMLDOC');
    renderer.renderToResponse(response);
}
```

Note: See the help topic [Using SuiteScript to Apply Advanced Templates to Non-Transaction Records](#) for a code sample and an explanation of how to use this function to print a record type that is not a transaction.

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjAssistant

See [nlobjAssistant](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjAssistantStep

See [nlobjAssistantStep](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjButton

See [nlobjButton](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjColumn

See [nlobjColumn](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjField

See [nlobjField](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjFieldGroup

See [nlobjFieldGroup](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjForm

See [nlobjForm](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjList

See [nlobjList](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjPortlet

See [nlobjPortlet](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjSubList

See [nlobjSubList](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjTab

See [nlobjTab](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

nlobjTemplateRenderer

See [nlobjTemplateRenderer](#) - defined in the section on [UI Objects](#).

[Back to UI Builder APIs](#) | [Back to SuiteScript Functions](#)

Application Navigation APIs

The following APIs let you define a navigation path for your users within NetSuite. Through these APIs you can redirect users to other standard or custom records within NetSuite. You can also direct them to custom Suitelets or other websites outside of NetSuite.

All APIs listed below are in alphabetical order.

- `nlapiRequestURL(url, postdata, headers, callback, httpMethod)`
- `nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)`
- `nlapiResolveURL(type, identifier, id, displayMode)`
- `nlapiSetRedirectURL(type, identifier, id, editmode, parameters)`
- `nlobjRequest`
- `nlobjResponse`

Note: If you have any hard-coded references to external URLs for Suitelets with the forms.netsuite.com domain, you must update these references. As of 2020.1, any external URL references with the old format will result in broken links. For access or redirection from another script to a Suitelet, the best practice is to use `url.resolveScript(options)` to discover the URL instead of hard-coding the URL. For more information about NetSuite domains, see the help topic [Understanding NetSuite URLs](#).

`nlapiRequestURL(url, postdata, headers, callback, httpMethod)`

Important: There are two “versions” of this API: a client-side version and a server-side version. When you execute this API in a server call, there is no **callback** parameter. Therefore, the function signature in a server-side call is `nlapiRequestURL(url, postdata, headers, httpMethod)`. When you execute this API in a client script, the function signature is `nlapiRequestURL(url, postdata, headers, callback, httpMethod)`.

Requests an HTTP(s) URL (internal or external). Note a timeout occurs if the initial connection takes > 5 seconds and/or the request takes > 45 to respond.

`nlapiRequestURL` automatically encodes binary content using base64 representation, since JavaScript is a character-based language with no support for binary types. This means you can take the contents returned and save them in the NetSuite file cabinet as a file or stream them directly to a response.

Important: NetSuite is now enforcing increased standards for hostname verification on all HTTPS requests. Hostname verification now ensures that a host's SSL certificate precisely matches the hostname. All scripts that attempt to make an HTTPS connection to an invalid host throw an exception and log an `SSS_INVALID_HOST_CERT` error.

Note: `nlapiRequestURL` supports TLS 1.2. SuiteScript 1.0 requests such as `nlapiRequestURL` and `nlapiRequestURLWithCredentials` will fail the handshake when they attempt to connect to servers that do not support TLS 1.2.

Also note that if you call `nlapiRequestURL`, passing in the header with a content type, NetSuite respects the following types:

- all text media types (types starting with "text/")
- "application/json"
- "application/vnd.maxmind.com-country+json"
- "application/xml"
- "application/soap+xml"
- "application/xhtml+xml"
- "application/atom+xml"

Otherwise, NetSuite will overwrite the content type with our default type as if the type had not been specified. NetSuite default types are:

- "text/xml; charset=UTF-8"
- "application/x-www-form-urlencoded; charset=UTF-8"

Additionally, `nlapiRequestURL` calls from server-side scripts do not include the current user's session information. This means you can only use this API to request Suitelets that are set to **available without login** using the external URL. Note that calls from client scripts do persist session information.

Usage metering allowed is 10 units. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **url** {string} [required] - The HTTP(s) URL being requested - (fully qualified unless NetSuite page)



Important: The url argument cannot include white space. If your URL includes white space, use the standard JavaScript function `encodeURIComponent` to encode the **url** argument. See http://www.w3schools.com/jsref/jsref_encodeuricomponent.asp and [Example 4](#) for additional information.

- **postdata** {string | object} [optional] - Body used for a POST request. It can either be an object of form parameters or a string. If set to **null**, then a GET request is used.



Note: If you specify DELETE for the **httpMethod** parameter, the **postdata** is ignored.

- **headers** {object} [optional] - An object of header and header value pairs. Be aware that certain headers cannot be set manually. If a script attempts to set a value for any of these headers, the value is discarded. These headers include the following: Connection, Content-Length, Host, Trailer, Transfer-Encoding, Upgrade, and Via.
- **callback** {function} [optional] - A callback function called when the request is completed (**client SuiteScript only**). IMPORTANT: There is NO **callback** parameter when you use `nlapiRequestURL` in a server-side call. In a server-side call, **httpMethod** becomes the fourth parameter for this API, as in: `nlapiRequestURL(url, postdata, headers, httpMethod)`.

If you specify a callback a client-side SuiteScript, the request is processed asynchronously, and after it is processed, the callback is called/invoked.

If you know your request may take a long time and you do not want to impair user experience, it is recommended that you set the callback function within `nlapiRequestURL` so that the request is processed asynchronously. If a callback function is specified, then the response will be passed, instead to the callback function, upon completion.

However, if validation is needed, `nlapiRequestURL` should run synchronously and the callback function should be omitted from `nlapiRequestURL`. For example:

```
var response = nlapiRequestURL(URL, postdata, header);
```



```
// callback function outside of the API call - will only execute after
// nlapiRequestURL has processed
function foo(response) {
    ...
}
```

- **httpMethod** {string} [optional] - Specify the appropriate http method to use for your integration. IMPORTANT: When using **nlapiRequestURL** in a server-side script, **httpMethod** becomes the fourth parameter. In other words, the function signature in a server-side script is **nlapiRequestURL(url, postdata, headers, httpMethod)**.

Supported http methods are HEAD, DELETE and PUT. This parameter allows for easier integration with external RESTful services using the standard REST functions. Note that if the **httpMethod** parameter is empty or not specified, this behavior is followed: the method is POST if **postdata** is not empty. The method is GET if it is.

Be aware that the **httpMethod** parameter overrides, so you can specify GET and specify postdata, NetSuite will do a GET and ignore the postdata.

Returns

- **nlobjResponse** object (**or** void if a callback function has been specified)



Important: NetSuite supports the same list of trusted third-party certificate authorities (CAs) as Microsoft. Click the following link for a list of these CAs: <http://social.technet.microsoft.com/wiki/contents/articles/31634.microsoft-trusted-root-certificate-program-participants-v-2016-april.aspx>

Throws

- **SSS_CONNECTION_CLOSED** (with "Connection Closed" message) — if the connection is closed because the server associated with the URL is unresponsive
- **SSS_CONNECTION_TIME_OUT** — if the initial connection exceeds the 5 second timeout period
- **SSS_INVALID_HOST_CERT** — if the client and server could not negotiate the desired level of security. The connection is no longer usable.
- **SSS_INVALID_URL** (with "Invalid URL — Connection Closed" message) — if the connection is closed due to an invalid URL, including those containing white space
- **SSS_REQUEST_TIME_EXCEEDED** — if the request exceeds the 45 second timeout period
- **SSS_UNKNOWN_HOST** — if the IP address of a host could not be determined.
- **SSS_UNSUPPORTED_ENCODING** — if the character encoding is not supported
- **SSS_REQUEST_LOOP_DETECTED** — if there is a potential infinite recursion problem

Example 1

Request an XML document from a server and also include a header.

```
var a = {"User-Agent-x": "SuiteScript-Call"};
var response = nlapiRequestURL('https://<accountID>.suitetalk.api.netsuite.com/wsd1/v1_2_0/netsuite.wsd1', null,
a);
var body = response.getBody();
var headers = response.getAllHeaders();
```

Example 2

Make an asynchronous request.

```
var a = {"User-Agent-x": "SuiteScript-Call"};
nlapiRequestURL('https://<accountID>.suitetalk.api.netsuite.com/wsdl/v1_2_0/netsuite.wsdl', null, a,
  handleResponse);
function handleResponse(response)
{
  var headers = response.getAllHeaders();
  var output = 'Code: '+response.getStatusCode()+'\n';
  output += 'Headers:\n';
  for (var i in headers)
    output += i + ': '+headers[i]+'\n';
  output += '\n\nBody:\n\n';
  output += response.getBody();
  alert( output );
}
```

Example 3

Make a request using a new browser window.

```
var a = {"User-Agent-x": "SuiteScript-Call"};
nlapiRequestURL('https://<accountID>.suitetalk.api.netsuite.com/wsdl/v1_2_0/netsuite.wsdl', null, a, null);
```

Example 4

Use the standard JavaScript function `encodeURIComponent(uri)` to encode an invalid URL that contains white space.

```
//Use encodeURIComponent when you want to encode a URL argument
var orderId = getOrderId(); //a dynamic generated value that could have space in it.
var customerId = getCustomerId(); //a dynamic generated value that could have space in it.
var url = 'www.netsuite.com/app/site/hosting/scriptlet.nl?script=187&deploy=1';
url += '&orderId=' + encodeURIComponent(orderId);
url += '&customerId=' + encodeURIComponent(customerId);
var response = nlapiRequestURL(url, null, null, null);
```

[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)

Supports the sending of credentials outside of NetSuite. This API securely accesses a handle to credentials that users specify in a NetSuite credential field.

Note: NetSuite credential fields can be added to Suitelets using the `nlobjForm.addCredentialField(id, label, website, scriptId, value, entityMatch, tab)` method.

Note a timeout occurs if the internal connections takes more than 5 seconds and/or the request takes more than 45 seconds to respond.

Also note that if you call `nlapiRequestURLWithCredentials`, passing in the header with a content type, NetSuite respects only the following two types:

- "application/json"
- "application/soap+xml"

Otherwise, NetSuite will overwrite the content type with our default type as if the type had not been specified. NetSuite default types are:

- "text/xml; charset=UTF-8"
- "application/x-www-form-urlencoded; charset=UTF-8"

You can Base64 any part of the request by wrapping any text in `$base64(<input text>)`. NetSuite will then Base64 encode the values in `<input text>`. This can be done in the value of a header, in the post body, or url. See [Example 2](#).

If you require additional encryption or encoding on the request string, you can pass an `nlobjCredentialBuilder` object to `nlapiRequestURLWithCredentials` in the url, postdata or headers argument. The `nlobjCredentialBuilder(string, domainString)` constructor takes in a user defined string, that can include an embedded globally unique string (GUID), and your URL's host name. `nlobjCredentialBuilder` includes six string transformation methods: two encryption methods for SHA-1 and SHA-256 encryption, two encoding methods for Base64 and UTF8 encoding, a character replacement method, and a string appending method. See [Example 3](#).


Usage metering allowed for this API is 10 units.

Supported Script Types


- User Event
- Scheduled Script
- Portlet
- Suitelet

Parameters

- **credentials** {array} [required — see Note] - List of credential handles. This API does not know where you have stored the data, it only knows the credentials to use by handle. Therefore, if you have multiple credentials for a single call, you need a list. The handles are 32 byte, globally unique strings (GUIDs).

 **Note:** If an `nlobjCredentialBuilder` object is passed in for the url, postdata or headers argument, you can pass in a null value for credentials.

- **url** {string | `nlobjCredentialBuilder` object} [required] - The HTTPS URL being requested - (fully qualified unless it is a NetSuite page).

 **Important:** If an `nlobjCredentialBuilder` object is passed in as the url, it must be passed in its original state (pre-encryption and pre-encoding); `nlapiRequestURLWithCredentials` cannot validate the url if it has been encrypted or encoded.

- **postdata** {string | `nlobjCredentialBuilder` object | hashtable} [optional] - Body used for a POST request. It can be a string, an `nlobjCredentialBuilder` object, an associative array of form parameter pairs, or an associative array of form parameter and `nlobjCredentialBuilder` object pairs. If set to `null`, then a GET request is used.

- **headers** {**nlobjCredentialBuilder** object | hashtable} [optional] - Can be an **nlobjCredentialBuilder** object, an associative array of header and header value pairs, or an associative array of header and **nlobjCredentialBuilder** object pairs. Be aware that certain headers cannot be set manually. If a script attempts to set a value for any of these headers, the value is discarded. These headers include the following: Connection, Content-Length, Host, Trailer, Transfer-Encoding, Upgrade, and Via.
- **httpsMethod** {string} [optional] - Specify the appropriate http method to use for your integration. Supported http methods are HEAD, DELETE and PUT. This parameter allows for easier integration with external RESTful services using the standard REST functions. Note that if the **httpsMethod** parameter is empty or not specified, this behavior is followed: the method is POST if **postData** is not empty. The method is GET if it is.

Be aware that the **httpsMethod** parameter overrides, so you can specify GET and specify postData, NetSuite does a GET and ignores the postData.

Returns

- **nlobjResponse** object

Since

- Version 2012.1

Example 1

The following shows a general process for creating credential fields and then, later, getting their handles and passing them on using **nlapiRequestURLWithCredentials**.

1. Two custom fields with username and passwords are added to a form:

```
var credField = form.addCredentialField('custpage_credname', 'password', null, valueFromCustomField, 'cert.merchante-solutions.com', 'customscript_usecredentialfield');
var usrfield = form.addCredentialField('custpage_username', 'username', null, valuefromusernamecustfield, 'cert.merchante-solutions.com', 'customscript_usecredentialfield');
```

2. During a beforeSubmit user event, obtain the values from credential fields and store them in two custom fields, which are not visible on the form:

```
var credValue = nlapiGetFieldValue('custpage_credname');
var username = nlapiGetFieldValue('custpage_username');

nlapiSetFieldValue('custentity_custompassword', credValue);
nlapiSetFieldValue('custentity_customusername', username);
```

3. Before using the credentials, copy them as a list in a variable. At this point, uname and pwd will contain the GUIDS (credentials handle):

```
var uname = rec.getFieldValue('custentity_customusername');
var pwd = rec.getFieldValue('custentity_custompassword');
var creds = [uname, pwd];
```

4. Use credentials in an external call:

```
var connect = nlapiRequestURLWithCredentials(creds, url);
```

Example 2

The following shows a general process for creating credential fields and then, later, getting their handles and passing them on using `nlapiRequestURLWithCredentials`. This example assumes that you already created the form and that the credentials are entered in a free-form text field.

1. Add credential fields during the beforeLoad event:

```
function doActionOnCredentialField()
{
    nlapiLogExecution('DEBUG', 'Inside the PageInit Script');
    var currentContext = nlapiGetContext();

    //Add a second tab to the form.
    var valuefromusernamecustfield = nlapiGetFieldValue('custbody1'); //fields from save credential script
    var valuefrompasswdcustfield = nlapiGetFieldValue('custbody2');

    nlapiLogExecution('DEBUG', 'Value of Custom Field custbody1: ', valuefromusernamecustfield);
    nlapiLogExecution('DEBUG', 'Value of Custom Field custbody2: ', valuefrompasswdcustfield);

    var myDomains = new Array('merchantesolutionstest', 'www.veritrans.co.jp', '<accountID>.app.netsuite.com');

    //refer script id of the use credential script here
    var usrfield = form.addCredentialField('custpage_username', 'username',
        myDomains, 'customscript19', valuefromusernamecustfield, false, null);
    var credField = form.addCredentialField('custpage_credname', 'password', myDomains, 'customscript19',
        valuefrompasswdcustfield, false, null);
}
```



Note: The addCredentialField function does not work for Suitelet that are available without login.

2. beforeSubmit – Save credentials:

```
function saveCredentialField()
{
    nlapiLogExecution('DEBUG', 'Inside the Save Credential Script');

    var currentContext = nlapiGetContext();
    var credValue = nlapiGetFieldValue('custpage_username');
    var username = nlapiGetFieldValue('custpage_credname');

    nlapiLogExecution('DEBUG', 'Credential Value: ', credValue);
    nlapiLogExecution('DEBUG', 'Username: ' + username);

    nlapiSetFieldValue('custbody1', credValue);
    nlapiSetFieldValue('custbody2', username);

    var valueFromCustomField = nlapiGetFieldValue('custbody_password');

    nlapiLogExecution('DEBUG', 'Value of Custom Field: ', valueFromCustomField);
}
```

3. afterSubmit – Use credentials:

```
function connectMES()
```

```

{
  nlapiLogExecution('DEBUG', 'Inside the connection script ');

  var currentContext = nlapiGetContext();
  var uname = nlapiGetFieldValue('custbody1');
  var pwd = nlapiGetFieldValue('custbody2');
  var url = 'https://<accountID>.app.netsuite.com/app/site/hosting/scriptlet.nl?script=20&deploy=1?';
  url = url + 'profile_id=' + '$base64((' + uname + '))';
  url = url + '&profile_key=' + '{'+pwd+'}';

  nlapiLogExecution('DEBUG', 'profile id - ' + uname);
  nlapiLogExecution('DEBUG', 'key - ' + pwd);
  nlapiLogExecution('DEBUG', 'url - ' + url);

  var creds = [uname,pwd];
  var connect = nlapiRequestURLWithCredentials(creds, url);
  var res = connect.body;

  nlapiLogExecution('DEBUG', 'response - ', connect.body);
}

```

Example 3

The following code instantiates an **nlobjCredentialBuilder** object and performs various modifications on it. A associative array of a form parameter and **nlobjCredentialBuilder** object pair is then passed into **nlapiRequestURLWithCredentials** as the postdata argument.

```

var _uname="user@company.com";
var _pwd = 'PASSWORD';

var creds = [_uname, _pwd];

var url = 'https://cert.merchante-solutions.com/mes-api/tridentApi?';
url = url + 'profile_id=' + '{'+_uname+'}';
url = url + '&profile_key=' + '{'+_pwd+'}';

var cbSha256 = new nlobjCredentialBuilder(url, 'cert.merchante-solutions.com').sha256();
var cbUtf8 = cbSha256.utf8();
var cbBase64 = cbUtf8.base64();
var cbReplace = cbBase64.replace('-', '*');

var a = nlapiRequestURLWithCredentials(creds ,url,{ 'CredentialBuilder' : cbReplace });

```


[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

nlapiResolveURL(type, identifier, id, displayMode)

Creates a URL on-the-fly by passing URL parameters from within your SuiteScript. For example, when creating a SuiteScript Portlet script, you may want to create and display the record URLs for each record returned in a search.

When creating the URL, you can use either the RECORD reference as retrieved in a search result or a known TASKLINK. Each page in NetSuite has a unique tasklink ID associated with it for a record type. Refer to the *SuiteScript Reference Guide* for a list of available NetSuite tasklinks.

This API is supported in client, user event, scheduled, portlet, Suitelet, and RESTlet scripts.

 **Note:** You can also discover the task ID for a NetSuite page by viewing the HTML page source and searching for the `n1PopupHelp` string. For example, this search might return `onclick="n1PopupHelp('LIST_SCRIPT','help')"`, where `LIST_SCRIPT` is the task ID.

Parameters

- **type** {string} [required] - The base type for this resource. These types include:
 - **RECORD** – Record Type
 - **TASKLINK** – Task Link
 - **SUITELET** – Suitelet
 - **RESTLET** – RESTlet
- **identifier** {string} [required] - The primary ID for this resource (recordType for RECORD, scriptId for SUITELET)
- **id** {string} [optional] - The secondary ID for this resource (recordId for RECORD, deploymentId for SUITELET).



Important: This argument is required if **type** has been set to **RECORD** and you are trying to resolve to a specific NetSuite record. In this scenario, you must set **id** to the ID of the target record.

- **displayMode** {string} [optional] - If the **type** argument is set to **RECORD**, set **displayMode** to either **VIEW** or **EDIT** to return a URL for the record in EDIT mode or VIEW mode. Note that even for RECORD calls, the **displayMode** argument is optional. The default value is VIEW.

If the **type** argument is set to **SUITELET** or **RESTLET**, set **displayMode** to **external** to return an external URL. Set **displayMode** to **internal**, or omit the argument, to return an internal URL. For Suitelets and RESTlets, **displayMode** automatically defaults to **internal**. **displayMode** also accepts **true** and **false** values, where **true** is used to return an external URL, and **false** is used to return an internal URL.

Returns

- Depending on the values specified for the **type** and **displayMode** arguments, returns URL string to an internal NetSuite resource **or** an external/internal URL to a Suitelet or RESTlet.

Throws

- SSS_INVALID_URL_CATEGORY
- SSS_CATEGORY_ARG_REQD
- SSS_INVALID_TASK_ID
- SSS_TASK_ID_REQD
- SSS_INVALID_INTERNAL_ID
- SSS_INVALID_EDITMODE_ARG

Example

The following lines of code show 5 different approaches for resolving to a record or Suitelet.

```
//resolve to a new Event record
```

```

var url_new_event = nlapiResolveURL('RECORD', 'calendarevent');

//resolve to a specific Event record page in view mode
var url_view_event = nlapiResolveURL('RECORD', 'calendarevent', 1000);

//resolve to a specific Event record in edit mode
var url_edit_event = nlapiResolveURL('RECORD', 'calendarevent', 1000, 'EDIT');

//resolve to a specified tasklink
var url_job_search = nlapiResolveURL('TASKLINK', 'SRCH_JOB');

//resolve to a specific Suitelet by specifying the Suitelet scriptId and deploymentId
var_url_servlet = nlapiResolveURL('SUITELET', 10, 5);

```

[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

nlapiSetRedirectURL(type, identifier, id, editmode, parameters)

Sets the redirect URL by resolving to a NetSuite resource. Note that all parameters must be prefixed with **custparam** otherwise an SSS_INVALID_ARG error will be thrown.

This API is supported in beforeLoad and synchronous afterSubmit user events; it is also supported in Suitelet scripts. Note that nlapiSetRedirectURL is ignored in beforeSubmit and asynchronous afterSubmit user events.

You can use **nlapiSetRedirectURL** to customize navigation within NetSuite. In a user event script, you can use **nlapiSetRedirectURL** to send the user to a NetSuite page based on a specific user event. For example, under certain conditions you may choose to redirect the user to another NetSuite page or custom Suitelet to complete a workflow.



Note: If you want to redirect a user to an external URL, you must use this function in a Suitelet and set the **type** parameter to EXTERNAL. See the documentation for the **type** parameter below.

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.



Important: To avoid **414 - Request URI Too large** or **502 — Bad Gateway** errors, limit the length of the URI to less than 2000 characters for all http requests. Different browsers and servers enforce different limits on URI length.

Parameters

- **type** {string} [required] - The base type for this resource. The types include:
 - **RECORD** : Record type - - Note that when you set **type** to RECORD, and the third param (**id**) to null, the redirection is to the “create new” record page, not an existing record page.
 - **TASKLINK** : Tasklink
 - **SUITELET** : Suitelet
 - **EXTERNAL** : The URL of a Suitelet that is available **externally** (for example, Suitelets that have been set to “Available without Login” on the Script Deployment page)



Important: The **EXTERNAL** value for **type** is only supported in Suitelets called with an external URL.

- **identifier** {string} [required] - The primary id for this resource (recordType for RECORD, scriptId for SUITELET, taskId for TASKLINK, url for EXTERNAL)
- **id** {string} [optional]- The secondary id for this resource (recordId for RECORD, deploymentId for SUITELET).



Important: This argument is **required** if *type* has been set to **RECORD** and you are trying to redirect to a specific NetSuite record. In the scenario, you must set **id** to the id of the target record.

- **editmode** {boolean **true** || **false**} [optional] - For RECORD calls, this 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.
- **parameters** {hashtable} [optional] - An associative array of additional URL parameters.



Important: All parameters must be prefixed with **custparam**.

Returns

- void

Throws

- SSS_INVALID_ARG
- SSS_INVALID_URL_CATEGORY
- SSS_CATEGORY_ARG_REQD
- SSS_INVALID_TASK_ID
- SSS_TASK_ID_REQD
- SSS_INVALID_INTERNAL_ID
- SSS_INVALID_EDITMODE_ARG

Example 1

The following example sets the redirect URL following the creation of an opportunity to a new task page. This script executes on an afterSubmit in a user event script.

```
if ( type == 'create' )
{
    var opportunity_id = nlapiGetRecordId();
    var params = new Array();
    params['opportunity'] = opportunity_id;
    nlapiSetRedirectURL('RECORD','task', null, null, params);
}
```

Example 2

This script sets the redirect URL to a newly created task record. Note that the record must exist and be submitted so the ID from the record can be used to set the redirect. This function is also executed on an afterSubmit in a user event script.

```
function redirectTaskRecord()
{
    var taskTitle = 'New Opportunity';
    var record = nlapiCreateRecord( 'task' );
    record.setFieldValue( 'title', taskTitle );
    id = nlapiSubmitRecord(record, true);
    nlapiSetRedirectURL( 'RECORD', 'task', id, false );
}
```

[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

nlobjRequest

See [nlobjRequest](#) - defined in the section on [Standard Objects](#).

[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

nlobjResponse

See [nlobjResponse](#) - defined in the section on [Standard Objects](#).

[Back to Application Navigation APIs](#) | [Back to SuiteScript Functions](#)

Date APIs

Use these APIs to manipulate standard JavaScript date and string objects.

In order to pass a JavaScript Date object as an argument into any SuiteScript 1.0 API, you need to convert it to the correct format by using [nlapiDateToString\(d, format\)](#) or [nlapiStringToDate\(str, format\)](#).

All APIs listed below are in alphabetical order.

- [nlapiAddDays\(d, days\)](#)
- [nlapiAddMonths\(d, months\)](#)
- [nlapiDateToString\(d, format\)](#)
- [nlapiStringToDate\(str, format\)](#)

nlapiAddDays(d, days)

Adds/subtracts a number of days to or from a date object

Parameters

- **d** {date} [required] - Date object
- **days** {int} [required] - Number of days being added to the date

Returns

- Date object corresponding to date that was passed in, plus the days you added or subtracted

[Back to Date APIs](#) | [Back to SuiteScript Functions](#)

nlapiAddMonths(d, months)

Adds/subtracts a number of months to or from a date object

Parameters

- **d** {date} [required] - Date object
- **months** {int} [required] - number of months being added to the date


Returns

- Date object corresponding to date that was passed in, plus the months you added or subtracted

[Back to Date APIs](#) | [Back to SuiteScript Functions](#)

nlapiDateToString(d, format)

Converts a Date object to a string, formats the string based on the format argument passed in, and then returns the formatted string.

 **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 current time in the Pacific Time Zone. Note that Daylight Savings Time does apply.

Parameters

- **d** {Date} [required] - Date object being converted into a string
- **format** {string} [optional] - Use one of the following arguments to determine the format of the returned string. Note that this parameter has no impact on time zone — see note above. If an argument is not passed in, the date format is used by default.
 - **date** — formats the string as a date, based on the Date Format selected in Set Preferences.
 - **timeofday** — formats the string as a time (hour and minutes), based on the Time Format selected in Set Preferences.
 - **datetime** — formats the string as a concatenation of date and time (hour and minutes), based on the Date Format and Time Format selected in Set Preferences
 - **datetimeetz** — formats the string as a concatenation of date and time (hour, minutes and seconds), based on the Date Format and Time Format selected in Set Preferences

Returns

- String format of the date that was passed

[Back to Date APIs](#) | [Back to SuiteScript Functions](#)

nlapiStringToDate(str, format)

Converts a string to a Date object, formats the date object based on the format argument passed in, and then returns the formatted date object. Be aware that leading zeroes in the month and day values are not supported.

Note: For client side scripts, the Date object returned is based on the user's system time. For server-side scripts, the Date object returned is based on the current time in the Pacific Time Zone. Note that Daylight Savings Time does apply.

Parameters

- **str** {string} [required] - String being converted to a Date.
- **format** {string} [optional] - Use one of the following arguments to indicate the format of the returned Date object. Note that this parameter has no impact on time zone — see note above.

Note: If you do not provide a format argument, your input string must not include seconds. Without a format argument, the returned Date object defaults to the date format.

- **datetime** — formats the Date object as a concatenation of date and time (hour and minutes), based on the Date Format and Time Format selected in Set Preferences. If you use this format type, your input string must not include seconds.
- **datetimez** — formats the Date object as a concatenation of date and time (hour, minutes and seconds), based on the Date Format and Time Format selected in Set Preferences. If you use this format type, your input string must include seconds.

Returns

- Date object. Returns NaN if date includes a leading zero.

Example

```
var myDate = nlapiStringToDate('8.5.2008'); // supported
var myDate = nlapiStringToDate('8/5/2008'); // supported

var myDate = nlapiStringToDate('08.5.2009'); // not supported
var myDate = nlapiStringToDate('08/5/2009'); // not supported

var myDate = nlapiStringToDate('8.05.2009'); // not supported
var myDate = nlapiStringToDate('8/05/2009'); // not supported
```

[Back to Date APIs](#) | [Back to SuiteScript Functions](#)

DateTime Time Zone APIs

Use these APIs in user event scripts to manipulate the default time zone set by NetSuite.

All APIs listed are in alphabetical order.

- [nlapiGetCurrentLineItemDateTimeValue\(type, fieldId, timeZone\)](#)
- [nlapiGetDateTimeValue\(fieldId, timeZone\)](#)
- [nlapiGetLineItemDateTimeValue\(type, fieldId, lineNum, timeZone\)](#)
- [nlapiSetCurrentLineItemDateTimeValue\(type, fieldId, dateTime, timeZone\)](#)
- [nlapiSetDateTimeValue\(fieldId, dateTime, timeZone\)](#)
- [nlapiSetLineItemDateTimeValue\(type, fieldId, lineNum, dateTime, timeZone\)](#)

nlapiGetCurrentLineItemDateTimeValue(type, fieldId, timeZone)

This API returns the value of a datetime field on the currently selected line of a sublist. If `timeZone` is passed in, the datetime value is converted to that time zone and then returned. If `timeZone` is not passed in, the datetime value is returned in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. This field ID must point to a datetime formatted field.
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- The string value of a Date/Time field on the currently selected line.

Throws

- `SSS_INVALID_ARG_TYPE`

Since

- Version 2013 Release 2

nlapiGetDateTimeValue(fieldId, timeZone)

This API returns the value of a datetime field. If `timeZone` is passed in, the datetime value is converted to that time zone and then returned. If `timeZone` is not passed in, the datetime value is returned in the default time zone.

Parameters

- **fieldId** {string} [required] — The internal field ID. This field ID must point to a datetime formatted field.
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- The string value of a datetime field.

Throws

- `SSS_INVALID_ARG_TYPE`

Since

- Version 2013 Release 2

Example

```
var tz = nlapiGetDateTimeValue('custrecord_datetimetz', 'America/Los_Angeles');
```

nlapiGetLineItemDateTimeValue(type, fieldId, lineNum, timeZone)

This API returns the value of a datetime field on a sublist. If `timeZone` is passed in, the datetime value is converted to that time zone and then returned. If `timeZone` is not passed in, the datetime value is returned in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **lineNum** {int} [required] — The line number for this field. Note the first line number on a sublist is 1 (not 0).
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- The string value of a datetime field on a sublist.

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

nlapiSetCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)

This API sets the value of a datetime field on the currently selected line of a sublist. If `timeZone` is passed in, the datetime value is converted to that time zone and then set. If `timeZone` is not passed in, the datetime value is set in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am|pm (for example, '09/25/2013 06:00:01 am').

- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

Example

```
nlapiselectNewItem('recmachcustrecord_childdatetime');
nlapisetCurrentLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzcol', '07/10/2013
06:00:01 am');
nlapicommitLineItem('recmachcustrecord_childdatetime');
```

nlapisetDateTimeValue(fieldId, dateTime, timeZone)

This API sets the value of a datetime field. If timeZone is passed in, the datetime value is converted from that time zone and then set. If timeZone is not passed in, the datetime value is set in the default time zone.

Parameters

- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am | pm (for example, '09/25/2013 06:00:01 am').
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

Example

```
nlapisetDateTimeValue('custrecord_datetimetz', '09/25/2013 06:00:01 am', 'Asia/Manila');
```

nlapisetLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)

This API sets the value of a datetime field on a sublist. If timeZone is passed in, the datetime value is converted to that time zone and then set. If timeZone is not passed in, the datetime value is set in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **lineNum** {int} [required] — The line number for this field. Note the first line number on a sublist is 1 (not 0).
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am|pm (for example, '09/25/2013 06:00:01 am').
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

Example

```
nlapisetLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzcol', 1, '09/25/2013 06:01:01 AM', 'Asia/Hong_Kong');
```

Currency APIs

Use these APIs to work with currency, as it pertains to your NetSuite account.

All APIs listed below are in alphabetical order.

- [nlapisExchangeRate\(sourceCurrency, targetCurrency, effectiveDate\)](#)
- [nlapisFormatCurrency\(str\)](#)

nlapiExchangeRate(sourceCurrency, targetCurrency, effectiveDate)

Use this API to get the exchange rate between two currencies based on a certain date. The exchange rate values you are getting are those that appear in the Exchange Rate column of the Currency Exchange Rates record (see figure).

Note: The Currency Exchange Rate record itself is not a scriptable record.

The usage metering allowed for this API is 10 units. This API is supported in all script types.

BASE CURRENCY	CURRENCY	EXCHANGE RATE	EFFECTIVE DATE
USA	British pound	1.50571501	8.1.2015
USA	Canadian Dollar	0.84672999	8.1.2015
USA	Euro	1.17691004	8.1.2015
USA	USA	1	31.8.2005
USA	Yen	0.00834547	8.1.2015

When using this API, the first currency (**sourceCurrency**) is the one to look up relative to the second (**targetCurrency**). The date (**effectiveDate**) is the rate in effect on that date. If there are multiple rates, it is the latest entry on that date.

For example, if you call `nlapiExchangeRate('GBP', 'USD', '04/22/2010')` and it returns '2', this means that if you were to enter an invoice on 4/22/10 for a GBP customer in your USD subsidiary, the rate would be 2.

Parameters

- **sourceCurrency** {string|int} [required] - The currency internal ID or symbol. For example, you can use either **1** (currency ID) or **USD** (currency symbol). If you have the Multiple Currencies feature enabled in your account, you can see all currency IDs and symbols by going to Lists > Accounting > Currencies.
- **targetCurrency** {string|int} [required] - The currency internal ID or symbol.
- **effectiveDate** {string|int} [optional] - If not supplied, then **effectiveDate** defaults to the current date.

Returns

- The exchange rate (as a decimal number) in the same precision that is displayed in the NetSuite UI.

Throws

- `SSS_INVALID_CURRENCY_ID` (if an invalid currency (from or to) is specified)

Since

- Version 2009.1

Example

This sample shows how to obtain the exchange rate between the Canadian dollar and the US dollar on March 17, 2009. The returned rate is applied against the Canadian dollar amount to obtain the amount in US dollars.

```
var canadianAmount = 100;
//specify source and target currencies as well as the exchange rate date
var rate = nlapiExchangeRate('CAD', 'USD', '03/17/2009');
var usdAmount = canadianAmount * rate;
```

[Back to Currency APIs](#) | [Back to SuiteScript Functions](#)

nlapiFormatCurrency(str)

Formats a String into a currency field value

Parameters

- **str** {string} [required] - String being formatted into currency

Returns

- String

[Back to Currency APIs](#) | [Back to SuiteScript Functions](#)

Encryption APIs

nlapiEncrypt(s, algorithm, key)

Encodes, encrypts, or obfuscates a clear text string.

Parameters

- **s** {string} [required] - The string to encode, obfuscate or encrypt.
- **algorithm** {string} [required] - The algorithm to use. See table for options.

Algorithm	Description
sha1	This option has been deprecated.
aes	Symmetric AES encryption
base64	Base-64 encoding
xor	Exclusive-OR obfuscation



Important: base64 encoding and XOR obfuscation are not forms of encryption.

- **key** {string} [optional] - The secret key that is used for AES encryption. Only applicable when using the aes algorithm. This string can be a 128-bit, 192-bit, or 256-bit hex key.

Returns

- String

[Back to Encryption APIs](#) | [Back to SuiteScript Functions](#)

XML APIs

Use these APIs when working with XML documents.

All APIs listed below are in alphabetical order.

- [nlapiEscapeXML\(text\)](#)
- [nlapiSelectNode\(node, xpath\)](#)
- [nlapiSelectNodes\(node, xpath\)](#)
- [nlapiSelectValue\(node, xpath\)](#)
- [nlapiSelectValues\(node, path\)](#)
- [nlapiStringToXML\(text\)](#)
- [nlapiValidateXML\(xmlDocument, schemaDocument, schemaFolderId\)](#)
- [nlapiXMLToString\(xml\)](#)
- [nlapiXMLToPDF\(xmlstring\)](#)

nlapiEscapeXML(text)

Prepares a String for use in XML by escaping XML markup (for example, angle brackets, quotation marks, and ampersands)

Parameters

- text** {string} [required] - String being escaped

Returns

- String

Example

In this line, **nlapiEscapeXML** is being used to escape special characters, such as an ampersand (&), that may appear in the names of items that are returned in an Item search. For the complete code sample, see [Example 2](#) in the API documentation for **nlapiXMLToPDF**.

```
strName += nlapiEscapeXML(searchresult.getValue( 'name' ) );
```

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectNode(node, xpath)

Selects a node from an XML document using an XPath expression

Parameters

- node** {node} [required] - XML node being queried

- **xpath** {string} [required] - XPath expression used to query node

Returns

- Node

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectNodes(node, xpath)

Selects an array of nodes from an XML document using an XPath expression

Parameters

- **node** {node} [required] - XML node being queried
- **xpath** {string} [required] - XPath expression used to query node

Returns

- Node[]

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectValue(node, xpath)

Selects a value from an XML document using an XPath expression

Parameters

- **node** {node} [required] - XML node being queried
- **xpath** {string} [required] - XPath expression used to query node

Returns

- String

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiSelectValues(node, path)

Selects an array of values from an XML document using an XPath expression

Parameters

- **node** {node} [required] - XML node being queried
- **path** {string} [required] - XPath expression used to query node

Returns

- String[]

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiStringToXML(text)

Parses a String into a W3C XML document. 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.

Parameters

- **text** {string} [required] - String being converted

Returns

- W3C Document object

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiValidateXML(xmlDocument, schemaDocument, schemaFolderId)

Validates a supplied XML document against a supplied XML Schema (XSD Document).



Important: nlapiValidateXML only validates XML Schema (XSD); validation of other XML schema languages is not supported.

The supplied XML Document and XSD Document must be passed in the form of a W3C Document object. Use [nlapiStringToXML\(text\)](#) to convert both documents before calling nlapiValidateXML. The location of your source XML Document and XDS Document does not matter; the validation is performed with the Document objects stored in memory.

XML Validation Output

If the validation is successful, **nlapiValidateXML** returns void. If the validation is not successful, **nlapiValidateXML** throws the error code `SSS_XML_DOES_NOT_CONFORM_TO_SCHEMA` and an `nlobjError` object containing error messages for the first 10 errors encountered. Use [nlapiLogExecution\(type, title, details\)](#) within a try catch statement to view these error messages; they are not automatically listed in the Execution Log.

```
try {
    nlapiValidateXML(xmlDocument, xsdDocument, '1234');
}
catch(e) {
    nlapiLogExecution('ERROR', 'XML Validation Failed: ' + e.getCode(), e.getDetails());
}
```

The log output will include up to three types of error messages: fatal errors, errors, and warnings.

```
Fatal Error: cvc-pattern-valid: Value '8812312319923' is not facet-valid with respect to
pattern '[0-9]{6}' for type 'orderidtype'.
Error: cvc-complex-type.3.2.2: Attribute 'bak' is not allowed to appear in element 'shiporder'.
Error: cvc-complex-type.3.2.2: Attribute 'ban' is not allowed to appear in element 'shiporder'.
Error: cvc-complex-type.3.2.2: Attribute 'binn' is not allowed to appear in element 'shiporder'.
Error: cvc-complex-type.3.2.2: Attribute 'dat' is not allowed to appear in element 'shiporder'.
Error: cvc-attribute.3: The value '8812312319923' of attribute 'orderid' on element 'shiporder'
is not valid with respect to its type, 'orderidtype'.
Error cvc-complex-type.2.2: Element 'option' must have no element [children], and the value
```

```

must be valid.
Error: cvc-complex-type.2.4.a: Invalid content was found starting with element 'property'. One
of '{property_id}' is expected.
Error: cvc-complex-type.3.2.2: Attribute 'title' is not allowed to appear in element
'shiporder'.
Warning: cvc-complex-type.3.2.2: Attribute 'expire' is not allowed to appear in element
'shiporder'.

```

Note that `nlapiLogExecution(type, title, details)` only logs warnings if errors are also logged. If `nlapiValidateXML(xmlDocument, schemaDocument, schemaFolderId)` encounters warnings and no errors, the validation passes.

Parameters

- **xmlDocument** {document} [required] — XML Document being validated.
- **schemaDocument** {document} [required] — XML Schema (in the form of an XSD Document) being validated against.
- **schemaFolderId** {string} [optional] — Only required if the passed XML Schema uses `<import>` or `<include>` tags that reference child schemas by file path (as opposed to references by URL. To use this parameter, upload the child schema(s) to a folder in the NetSuite file cabinet. Then pass the folder internal ID as the **schemaFolderId** argument. Note that SuiteScript ignores this argument if it is passed, but not needed.

Returns

- Void

Throws

- `SSS_XML_DOES_NOT_CONFORM_TO_SCHEMA` — Thrown when the validation fails. See [XML Validation Output](#) for additional information.
- `SSS_XML_SCHEMA_MISSING_DEPENDENCY_FOLDER_ID` — Thrown when an invalid **schemaFolderId** argument is passed; also thrown when **schemaFolderId** is required but missing.

Since

- Version 2014 Release 1

Example

```

//load an XML document from the file cabinet
var xmlFile = nlapiLoadFile('1234');
var xmlDocument = nlapiStringToXML(xmlFile.getValue());

//load an XSD document from the file cabinet
var xsdFile = nlapiLoadFile('4321');
var xsdDocument = nlapiStringToXML(xsdFile.getValue());

//validate that the XML document conforms to the schema
try {
    nlapiValidateXML(xmlDocument, xsdDocument, '1234');
}
catch(e) {
    nlapiLogExecution('ERROR', 'XML Validation Failed: ' + e.getCode(), e.getDetails());
}

```

```
nlapilogExecution('ERROR', 'XML Validation Succeeded', xmlFile.getName());
```

nlapiXMLToString(xml)

Converts (serializes) an XML document into a String. This API is useful if you want to serialize and store a Document in a custom field (for example).

Parameters

- **xml** {W3C Document} [required] - XML document being serialized


Returns

- String

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

nlapiXMLToPDF(xmlstring)

Use this API in conjunction with the Big Faceless Report Generator built by Big Faceless Organization (BFO). The BFO Report Generator is a third-party library used for converting XML to PDF documents. BFO is used in NetSuite. For version details, see the help topic [Third-Party Notices and Licenses](#). Using **nlapiXMLToPDF** in combination with the BFO report library, SuiteScript developers can now generate PDF reports from Suitelets.

 **Note:** SuiteScript developers do not need to install any BFO-related files or components to use the Report Generator functionality.

The **nlapiXMLToPDF** API passes XML to the BFO tag library (which is stored by NetSuite), and returns a PDF [nlobjFile](#) object. Note that there is a **10MB** limitation to the size of the file that can be created.

The following list includes some of the output styles that can be generated using **nlapiXMLToPDF** and BFO tags:

- Consolidated data from multiple transactions into one (for example, a virtual consolidated invoice)
- Highly tailored transaction output with images
- Product labels with bar codes
- Pallet labels with bar codes (custom records)
- Custom-formatted product catalogs with images
- Proposals

Escaping control characters is not supported.

For details on BFO, available tags, and BFO examples, see the following links:

- <http://faceless.org/products/report/docs/userguide.pdf>
- <http://faceless.org/products/report/docs/tags/>

For more information about advanced PDF and HTML templates, see the help topic [Advanced PDF/HTML Templates](#).

Parameters

- **xmlstring** {string} [required] – XML

Returns

- PDF [nlobjFile](#) object

Throws

- Error: ERROR_PARSING_XML (thrown as a user error when XML is badly formed)

Since

- Version 2009.1

Example 1

This sample shows how to generate a PDF from a Suitelet. The output is a PDF that reads Hello World! See also, [Working with BFO \(the Basics\)](#).

```
function helloWorld()
{
    var xml = "<?xml version='1.0'>\n<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\n>\n<pdf>\n<body font-size='18'\nHello World!\n</body>\n</pdf>";
    var file = nlapiXMLToPDF( xml );
    response.setContentType( 'PDF', 'helloworld.pdf' );
    response.write( file.getValue() );
}
```

Example 2

This sample shows how to create a PDF of a pricing list. All data for the pricing list is pulled from NetSuite, organized into tables, and then transformed into a PDF.

```
function priceListPDF(request, response)
{
    // set search filters for pricing list search
    var filters = new Array();

    // against pricing lists, search for a specific customer
    filters [0] = new nlobjSearchFilter('customer', 'pricing', 'is', '121');

    // against pricing lists, look for lists that have currency defined as USA
    filters [1] = new nlobjSearchFilter('currency', 'pricing', 'is', '1');

    // set search return columns for pricing list search
    var columns = new Array();
    columns[0] = new nlobjSearchColumn('pricelevel', 'pricing');
    columns[1] = new nlobjSearchColumn('unitprice', 'pricing');
    columns[2] = new nlobjSearchColumn('name');

    // when doing a pricing list search you must specify 'item' as the search type
    var searchresults = nlapiSearchRecord('item', null, null, columns);

    // create a table to present the results of the search
    var strName = "<table>";

    // iterate through the results
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
```



```

{
    searchresult = searchresults[ i ];
    strName += "<tr><td>";
    // note the use of nlapiEscapeXML to escape any special characters,
    // such as an ampersand (&) in any of the item names
    strName += nlapiEscapeXML(searchresult.getValue( 'name' ) );
    strName += "</td>";
    strName += "<td>";
    strName += searchresult.getValue( 'unitprice', 'pricing' );
    strName += "</td>";
    strName += "<td>";
    strName += "<barcode codetype=\"code128\" showtext=\"true\" value=\""; strName += searchresult.getValue(
'unitprice', 'pricing' ); strName += "\"/>";
    strName += "</td></tr>";
}
strName += "</table>";

// build up BFO-compliant XML using well-formed HTML
var xml = "<?xml version=\"1.0\"?>\n<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//
report\" \"report-1.1.dtd\">\n";
xml += "<pdf>\n<body font-size=\"12\">\n<h3>My Pricing List</h3>\n";
xml += "<p></p>";
xml += strName;
xml += "</body>\n</pdf>";


// run the BFO library to convert the xml document to a PDF
var file = nlapiXMLToPDF( xml );

// set content type, file name, and content-disposition (inline means display in browser)
response.setContentType('PDF','Pricing List.pdf', 'inline');

// write response to the client
response.write( file.getValue() );
}

```

My Pricing List

Cleaning	24.00	
Replacing of Dental Drill Head	125.00	
Serving	18.00	
Cotton Swabs	7.50	

Example 3

For NetSuite customers who want to print a PDF that includes Cyrillic characters (Russian text), this sample shows how to point to a Russian font set hosted by NetSuite. To print Russian text, you must include the <link> tag within the <head>. The path in your <link> tag must be the exact path that is specified here in this sample.

```
function main(Request, Response)
{
    var xml = "<?xml version='1.0' encoding='UTF-8'?'>\n" +
        "<!DOCTYPE pdf PUBLIC \"-//big.faceless.org//report\" \"report-1.1.dtd\">\n" +
        "<pdf lang='ru-RU' xml:lang='ru-RU'>\n" +
        "    <head>\n" +
        "        <link name='russianfont' type='font' subtype='opentype' " + "src='NetSuiteFonts/verdana" +
        ".ttf\" " + "src-bold='NetSuiteFonts/verdanab.ttf\" " + "src-italic='NetSuiteFonts/verdanai.ttf\" " + "src-" +
        "bolditalic='NetSuiteFonts/verdanabi.ttf\" " + "bytes='2'/'>\n" +
        "    </head>\n" +
        "    <body font-family='russianfont' font-size='18'>\n" +
        "        <p>Russian: Русский текст</p>\n" +
        "        <p>Russian Italic: <i>Русский текст</i></p>\n" +
        "        <p>Russian Bold: <b>Русский текст</b></p>\n" +
        "        <p>Russian Bold Italic: <b><i>Русский текст</i></b></p>\n" +
        "    </body>\n" +
        "</pdf>";
    var file = nlapiXMLToPDF( xml );
    Response.setContentType( 'PDF', 'helloworld.pdf', 'inline' );
    Response.write( file.getValue() );
}
```

Working with BFO (the Basics)

For convenience, the following basic coding details regarding BFO are here for SuiteScript developers. For more detailed explanations, see the section called “Creating the XML - A Simple Example” in the BFO User Guide (<http://faceless.org/products/report/docs/userguide.pdf>).

1. The XML declaration `<?xml version="1.0"?>` must always be included as the very first line of the file.
2. The DOCTYPE declaration tells the XML parser which DTD to use to validate the XML against.
3. The top level element of the XML document must always be `<pdf>`.
4. Like HTML, the document consists of a “head”, containing information about the document, and a “body” containing the contents of the document.
5. In XML an element must always be “closed” - this means that `<pdf>` must always be matched by `</pdf>`, `` by `` and so on. When an element has no content, like `
`, `` or `<meta>`, it may close itself.
6. The `<body>` element can have some attributes set - background-color and font-size. In XML, every attribute value must be quoted - this can be frustrating for HTML authors used to typing `<table width=100%>`.

[Back to XML APIs](#) | [Back to SuiteScript Functions](#)

File APIs

Use these APIs to work with files that currently exist in the NetSuite file cabinet. These APIs can also be used to create files to load into NetSuite or to send as attachments in email.


All APIs listed below are in alphabetical order.

- `nlapiCreateFile(name, type, contents)`
- `nlapiDeleteFile(id)`

- [nlapiLoadFile\(id\)](#)
- [nlapiSubmitFile\(file\)](#)
- [nlobjFile](#)

nlapiCreateFile(name, type, contents)


Instantiates and returns an [nlobjFile](#) object. The file object can be used as an email or fax attachment. The file object can also be saved to the file cabinet using [nlapiSubmitFile\(file\)](#).

 **Note:** There is a 10MB limitation to the size of the document that can be created using this API.

The **nlapiCreateFile** API can also be used for streaming to clients (via Suitelets). For streaming or attaching binary content, you can call the following. Note that each of these APIs can load or generate binary content, provided that the **contents** argument is **base-64** encoded.

- [nlapiLoadFile\(id\)](#)
- [nlapiPrintRecord\(type, id, mode, properties\)](#)
- [nlapiMergeRecord\(id, baseType, baseId, altType, altId, fields\)](#)

This API is supported in user event, scheduled, portlet, mass update, and Suitelet scripts.

 **Important:** Be aware that the **nlapiCreateFile** function does not support the creation of non-text file types such as PDFs, unless the **contents** argument is base-64 encoded.

Parameters

- **name** {string} [required] - The file name and extension.
- **type** {string} [required] - The file type. For a list of supported file types, see the help topic [SuiteScript 1.0 Supported File Types](#) in the NetSuite Help Center. Note that when specifying the *type* for an on demand email or fax attachment, only non-binary types are supported (for example, PLAINTEXT, HTMLDOC, XMLDOC), **unless** the **contents** argument is base-64 encoded.
- **contents** {string} [required] - The contents of the file.

Returns

- An [nlobjFile](#) object

Since

- Version 2008.1

Example 1

This example shows how to create a basic text file to use as an email attachment. Note that after it is created, the file object will not be stored in the file cabinet.

```
function sendAttachment()
{
    var newAttachment = nlapiCreateFile('helloworld.txt', 'PLAINTEXT', 'Hello World\nHello World');

    var newEmail = nlapiSendEmail(210, 'kwolfe@netsuite.com', 'Sample email and attachment',
        'Please see the attached file', null, null, null, newAttachment);
}
```

Example 2

This example shows how to turn a file merge into a PDF document object. The PDF can then be used as an email attachment.

```
var pdfcontents = nlapiMergeRecord(....)
var fileObj = nlapiCreateFile('mypdf.pdf', 'PDF', pdfcontents)
```

[Back to File APIs](#) | [Back to SuiteScript Functions](#)

nlapiDeleteFile(id)

Deletes a file and returns the internal ID of the file that was deleted. Usage metering allowed for this function is 20 units. This API is supported in user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **id** {int} [required] - The internal ID for the file you want to delete

Returns

- The internal ID for the file that was deleted as an integer

Since


- Version 2009.1

[Back to File APIs](#) | [Back to SuiteScript Functions](#)

nlapiLoadFile(id)

Loads a file from the NetSuite file cabinet (using the file's internal ID or path). Returns an [nlobjFile](#) object that encapsulates the file's metadata (name and type) and contents in the form of a String (base-64 encoded if the file's type is binary). The script context must have privileges to the file (based on folder permissions), and the file cannot be a hidden (bundled) file.

Usage metering allowed for nlapiLoadFile is 10 units. This API is supported in server-side scripts.

 **Note:** `nlapiLoadFile` can load `nlobjFile` objects of any size, as long as the file size is permitted by the file cabinet.

Parameters

- **id** {string | int} [required] - The internal id of the file in the file cabinet. Can also be a relative path to the file in the file cabinet (for example: SuiteScript/myfile.js).

Returns

- An [nlobjFile](#) object

Example


This example shows how to load a jpeg that is currently in the Images folder in the File Cabinet. The script returns the file as a NetSuite [nlobjFile](#) object, and you can use nlobjFile methods to interact with the file.

```
function logEvent(type)
{
    try {
        var f = nlapiLoadFile('Images/logo_goat55.jpg');
        nlapiLogExecution('AUDIT', 'Event', 'Type: ' + type + ' File: ' + f.getId());
    }
    catch(err) {
        nlapiLogExecution('AUDIT', 'Event', 'An error occurred when trying to load the file.');
```

[Back to File APIs](#) | [Back to SuiteScript Functions](#)

nlapiSubmitFile(file)

Submits a file and returns the internal ID to the file that was added to (or updated in) the NetSuite file cabinet. Note that if a file with the same name exists in the folder that this file is added to, then that file will be updated.

 **Note:** nlapiSubmitFile can submit nlobjFile objects of any size, as long as the file size is permitted by the file cabinet .

Usage metering allowed for this function is 20 units. This API is supported in user event, scheduled, portlet, and Suitelet scripts.

Parameters

- *file* {[nlobjFile](#)} [required] - The **nlobjFile** that will be updated

Returns

- The integer value of the file ID.

Since

- Version 2009.1

Example

- See the code sample in [Uploading Files to the File Cabinet Using SuiteScript](#).

[Back to File APIs](#) | [Back to SuiteScript Functions](#)

nlobjFile

See [nlobjFile](#) - defined in the section on [Standard Objects](#).

[Back to File APIs](#) | [Back to SuiteScript Functions](#)

Error Handling APIs

All APIs listed below are in alphabetical order.

- [nlapiCreateError\(code, details, suppressNotification\)](#)
- [nlobjError](#)

nlapiCreateError(code, details, suppressNotification)

Creates an [nlobjError](#) (complete with stacktrace) that can be thrown to abort script execution. This API is supported in user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **code** {string} [required] - A user-defined error code
- **details** {string} [required] - The error details
- **suppressNotification** {boolean **true** | **false**} [optional] - If not set, defaults to false and an email notification with error details is sent after script execution. If set to **true**, the error email notification is suppressed.

Returns

- An [nlobjError](#) object
- [Back to Error Handling APIs](#) | [Back to SuiteScript Functions](#)

nlobjError

See [nlobjError](#) - defined in the section on [Standard Objects](#).

[Back to Error Handling APIs](#) | [Back to SuiteScript Functions](#)

Communication APIs

Use these APIs to communicate to external systems from within NetSuite.

All APIs listed below are in alphabetical order.

- [nlapiSendCampaignEmail\(campaigneventid, recipientid\)](#)
- [nlapiSendEmail\(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo\)](#)
- [nlapiSendFax\(author, recipient, subject, body, records, attachments\)](#)
- [nlapiOutboundSSO\(id\)](#)

nlapiSendCampaignEmail(campaigneventid, recipientid)

Use this function to send a single on-demand campaign email to a specified recipient and return a campaign response ID to track the email. This function works in conjunction with the Lead Nurturing (campaigndrip) sublist only; it does not work with the E-mail (campaignemail) sublist.

Campaign Email Volume provisioning is used for the account. 10 units of usage metering is allowed. This API is supported in user event, scheduled, Suitelet, mass update, and workflow action scripts.

Parameters

- **campaigneventid** {int} [required] - The internal ID of the campaign event. The campaign must be of type **campaigndrip**, which is referred to as Lead Nurturing in the UI.
- **recipientid** {int} [required] - The internal ID of the recipient. Note that the recipients must have an email. Values can be:
 - A single string value of the recipient's email address
 - A single integer value of the recipient's internal ID
 - An array of strings of the recipients' email addresses. The primary recipient must be the first element in the array of strings.
 - An array of integers of the recipients' internal IDs. The primary recipient must be the first element in the array of integers.
 - A mixed array of and strings (recipient email addresses) and integers (recipient internal IDs). The primary recipient must be the first element in the mixed array of strings and integers.

Returns

- A campaign response ID (tracking code) as an integer, or -1 if the send fails.

Since

- Version 2010.1

Example

This sample shows how to create a new campaign event and email the event to a specified recipient. After the email is sent, the sender can use the campaign response ID that is returned for tracking purposes.

```
// Create the new campaign record in dynamic mode so all field values can be dynamically sourced.
// For information on dynamic scripting, see Working with Records in Dynamic Mode.
var campaign1 = nlapiCreateRecord('campaign', {recordmode: 'dynamic'});
campaign1.setFieldValue('title', 'Sample Lead Nurturing Campaign');

//Set values on the Lead Nurturing (campaigndrip) sublist
campaign1.selectNewLineItem('campaigndrip');

// 4 is a sample ID representing an existing marketing campaign
campaign1.setCurrentLineItemValue('campaigndrip', 'template', 4);
campaign1.setCurrentLineItemValue('campaigndrip', 'title', 'Sample Lead Nurturing Event');

// 1 is a sample ID representing an existing subscription
campaign1.setCurrentLineItemValue('campaigndrip', 'subscription', 1);

// 2 is a sample ID representing an existing channel
campaign1.setCurrentLineItemValue('campaigndrip', 'channel', 2);

// 1 is a sample ID representing an existing promocode
campaign1.setCurrentLineItemValue('campaigndrip', 'promocode', 1);
campaign1.commitLineItem('campaigndrip');

// Submit the record
var campaign1Key = nlapiSubmitRecord(campaign1);
```

```
// Load the campaign record you just created. Determine the internal ID of the campaign event
// to the variable campaign2_campaigndrip_internalid_1.
var campaign2 = nlapiLoadRecord('campaign', campaign1Key, {recordmode: 'dynamic'});
var campaign2_campaigndrip_internalid_1 = campaign2.getLineItemValue('campaigndrip', 'internalid', 1);

// 142 is a sample ID representing the ID of a recipient with a valid email address
var campaignResponseId = nlapiSendCampaignEmail(campaign2_campaigndrip_internalid_1, 142);
```

[Back to Communication APIs](#) | [Back to SuiteScript Functions](#)

nlapiSendEmail(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo)

nlapiSendEmail sends and records outgoing email to an individual or to a group of individuals. You can use **nlapiSendEmail** in the following ways:

- To send bulk email.
- To send important email, for which you need bounceback notifications when the email is not successfully delivered. To do this, set **notifySenderOnBounce** to true. Note that when this parameter is used, the maximum number of total recipients (recipient + cc + bcc) allowed is 10. In addition, the governance is increased to 20 usage units.
- To attach emails to custom records. To do this, reference the custom record by either its **internalId** or **scriptId**. You can send multiple attachments of any media type with this function. Email messages have a 15MB size limit. The total size of the message plus any attachments must be 15MB or less. The size of any individual attachment may not exceed 10MB.
- To send an email message from one address and to receive replies at another address. To do this, use the **replyTo** parameter.

Note: This API normally uses a bulk email server to send emails. When **notifySenderOnBounce** is set to true though, **nlapiSendEmail** uses a different, transactional, email server with a higher successful delivery rate. If you need to increase the successful delivery rate of an email, set **notifySenderOnBounce** to true even if you do not need bounceback notifications.

You can use NetSuite email templates to construct the body of the email using a set of APIs supporting scriptable templates. For information on these APIs, see [nlapiCreateEmailMerger\(templateId\)](#). Version 2014 Release 1 introduced scriptable templates as a replacement for CRMSDK templates. CRMSDK templates were deprecated with Version 2015 Release 1 and will no longer be supported as of Version 2016 Release 1. To facilitate this final transition to scriptable templates, Version 2015 Release 1 also deprecated the SuiteScript function **nlapiMergeRecord**, used to perform mail merges with CRMSDK templates. This function will no longer be supported as of Version 2016 Release 1.


Note: If your body argument contains XML tags and you want SuiteScript to format the body as plain text, wrap the XML in an HTML `<pre>` tag (`<pre> XML goes here </pre>`). The `<pre>` tag tells the email client that the body is pre-formatted and instructs the client to ignore any control characters. When the email is opened, the XML displays as plain XML source.


This API is supported in all client and server-side script types. When **notifySenderOnBounce** is not used, the governance for this function is 10 usage units. When **notifySenderOnBounce** is set to true, the governance for **nlapiSendEmail** increases from 10 to 20 usage units.

Parameters


There can be multiple related records, but only one of each parameter (each parameter represents applicable record types).

- **author** {int} [required] - The internalId of an employee record (this is the sender). To get the internal ID for an employee, go to Lists > Employees > Employees (you must have admin access to the account to access the Employees list page). The employee's ID will appear in the Internal ID column on the list page. Note that you must have the **Show Internal IDs** preference enabled in your account. To enable this preference, go to Home > Set Preferences > General tab > under Defaults > click **Show Internal IDs** > click **Save**.
- **recipient** {string | int} [required] - Set one of the following for this parameter:
 - A single external email address
 - An array of email addresses
 - A list of external addresses (comma separated)


 **Note:** If multiple recipients are passed, only the first recipient displays on the Communication tab (under the Recipient column). This is due to the design of the UI. To view all recipients, click **View** to open the Message record. The complete list of recipients displays on the Recipients tab.

 **Important:** When **notifySenderOnBounce** is true, the maximum number of total recipients (recipient + cc + bcc) allowed is 10.

- The internal ID of a single entity in NetSuite. Note that if the internal ID of the recipient entity record is used, the email message is automatically attached to the entity record.
- **subject** {string} [required] - Subject of the outgoing mail. A JavaScript exception is thrown if this argument is left blank, set to **null**, or set to an empty string.
- **body** {string | nlobjFile[]} object returned from nlapiMergeRecord} [required] - Body of the outgoing email. A JavaScript exception is thrown if this argument is left blank, set to **null**, or set to an empty string.
- **cc** {string | string[]} [optional] - An array of email addresses or a single email address to copy

 **Important:** When **notifySenderOnBounce** is true, the maximum number of total recipients (recipient + cc + bcc) allowed is 10.

- **bcc** {string | string[]} [optional] - An array of email addresses or a single email address to blind copy.

 **Important:** When **notifySenderOnBounce** is true, the maximum number of total recipients (recipient + cc + bcc) allowed is 10.

- **records** {hashtable} [optional] - An associative array of internal records to associate/attach this email with. The following table lists valid keys -> values.

Key	Value (examples)
transaction (use for transaction and opportunity record types)	records[' transaction '] = ' 1000 ';
activity (use for Case and Campaign record types)	records[' activity '] = ' 50 ';
entity	records[' entity '] = ' 555 ';

Key	Value (examples)
(use for all Entity record types, for example, customer, contact, etc.)	
record (custom record internalId - for custom records you must also specify both the record ID and the record type ID)	records['record'] = '3';
recordtype (custom recordtype internalId or scriptId)	records[' recordtype '] = ' customrecord11 ';

- **attachments** {nlobjFile | nlobjFile[]} [optional] - A single [nlobjFile](#) object - **or** - an array of nlobjFile objects to attach to outgoing email (**not** supported in Client SuiteScript).



Important: The total size of the message plus any attachments must be 15MB or less. The size of any individual attachment may not exceed 10MB.

- **notifySenderOnBounce** {Boolean true | false} [optional] — A value of true causes bounceback notifications to be sent to the original sender for each supplied recipient. Note that bounceback notification support is dependent upon the recipient's email server settings.



Important: When **notifySenderOnBounce** is true, the maximum number of total recipients (recipient + cc + bcc) allowed is 10. In addition, the governance for **nlapiSendEmail** increases from 10 to 20 usage units per execution.

- **internalOnly** {Boolean true | false} [optional] — A value of true sets a new message record as internal only. When a message record is set to internal only, customers do not see the message from the customer center.
- **replyTo** {string} [optional] — The email address that appears in the reply-to header when an email is sent out. If the recipient replies to the email, the value passed to **replyTo** is prepopulated in the To: field of the recipient's response.

Set one of the following for this parameter:

- A single external email address
- A generic email address created by the plug-in. For more information about the Email Capture Plug-in, see the help topic [Email Capture Plug-in Overview](#).

Returns

- void

Throws

- **SSS_AUTHOR_MUST_BE_EMPLOYEE** — Thrown when an invalid internal ID is passed for the author parameter.
- **SSS_AUTHOR_REQD** — Thrown when the author argument is left blank, set to null, or set to an empty string.
- **SSS_File_CONTENT_SIZE_EXCEEDED** — Thrown when an attachment exceeds the 10MB file size limit.
- **SSS_INVALID_BCC_EMAIL** — Thrown when an invalid email address is passed for the bcc parameter.
- **SSS_INVALID_CC_EMAIL** — Thrown when an invalid email address is passed for the cc parameter.
- **SSS_INVALID_RECIPIENT_ID** — Thrown when an invalid internal ID is passed for the recipient parameter.
- **SSS_INVALID_REPLYTO_EMAIL** — Thrown when an invalid email address is passed for the replyTo parameter.

- `SSS_INVALID_TO_EMAIL` — Thrown when an invalid email address is passed for the recipient parameter.
- `SSS_MAXIMUM_NUMBER_RECIPIENTS_EXCEEDED` — Thrown when `notifySenderOnBounce` is true and the total number of recipients (recipient + cc + bcc) exceeds 10.
- `SSS_MISSING_REQD_ARG` — Thrown when a required argument is left blank, set to null, or set to an empty string.
- `SSS_RECIPIENT_REQD` — Thrown when the recipient argument is left blank, set to null, or set to an empty string.

Example 1

```
// Merge, send, and associate an email with an opportunity record (id=1000)
function testMergeAndSendEmail()
{
    var records = new Object();
    records['transaction'] = '1000';

    var emailBody = nlapiMergeRecord( 25, 'customer', '100' ).getValue();
    nlapiSendEmail( -5, 'customer@customer.com', 'Promotion Notification',
        emailBody, null, null, records );
}
```

Example 2

This example shows how to send an email that includes an attachment.

```
var newAttachment = nlapiLoadFile(67);

nlapiSendEmail(author, recipient, subject, body, null, null, records, newAttachment);
```

Example 3

This example shows how to associate an outgoing email with a custom record.

```
var records = new Object();
records['recordtype'] = InternalIdOfCustomRecordType; // for example 55
records['record'] = InternalIdOfCustomRecord;

nlapiSendEmail(1, custemail, emailsubj, emailtext, null, null, records);
```

Example 4

This example shows the `notifySenderOnBounce` argument set to `true`. The original sender receives a bounceback notification for each recipient email not successfully delivered.

```
nlapiSendEmail(1, ['msample@netsuite.com', 'jdoe@netsuite.com'],
    'hello world', 'your order has been completed',
    ['sales@netsuite.com', account-management@netsuite.com'], mySalesOrder, myPdf, true);
```

Example 5

This example shows how to send an email from the original sender, to a recipient, `customer@customer.com`. The reply-to field of the email will be set to `accounts@netsuite.com`.

```
nlapiSendEmail(1, 'customer@customer.com',
               'Invoice Receipt', 'your order has been completed',
               null, null, null, null, true, null, 'accounts@netsuite.com');
```

[Back to Communication APIs](#) | [Back to SuiteScript Functions](#)

nlapiSendFax(author, recipient, subject, body, records, attachments)

Sends and records an outgoing fax using the fax settings already defined in the user's account. This API is supported in client, user event, scheduled, portlet, and Suitelet scripts.

Parameters

- **author** {int} [required] - InternalId of an employee record (this is the sender)
- **recipient** {string} [required] - InternalId of the recipient entity -or- a free-form fax (if set to an internalId the fax will be saved)
- **subject** {string} [required] - Subject of the outgoing fax
- **body** {string} [optional] - Body of the outgoing fax
- **records** {hashtable} [optional] - Name/value pairs of internal records to associate this fax with (if set, fax will be saved)
 - transaction - transaction/opportunity internalid
 - activity - case/campaign internalid
 - entity - entity internalid
 - record - custom record internalId
 - recordtype - custom recordType internalId (or script id)
- **attachments** {nlobjFile} [optional] - array of [nlobjFile](#) objects or a single [nlobjFile](#) object to attach to outgoing fax (not supported in Client SuiteScript)

Returns

- void

Since

- Version 2008.1

Example

```
// Merge, send, and associate a fax with an customer record (id=1000)
function testMergeAndSendFax()
{
    var records = new Object();
    records['entity'] = '1000';

    var faxBody = nlapiMergeRecord( 25, 'customer', '100' ).getValue();
    nlapiSendFax( -5, '650.555.4455', 'Promotion Notification', faxBody, records );
}
```

```
}
```

[Back to Communication APIs](#) | [Back to SuiteScript Functions](#)

nlapiOutboundSSO(id)

Use this API to generate a new OAuth token for a user. Currently this API can be called from portlet scripts, user event scripts, and Suitelets **only**. This API consumes 20 usage units per call.

Note that you must have the SuiteSignOn feature enabled in your account before you can use SuiteSignOn functionality. (To enable these features, go to Setup > Company > Enable Features. On the SuiteCloud tab, select the web services check box and the SuiteSignOn check box, then click Save.)



Important: For complete details on the SuiteSignOn feature, see the *SuiteSignOn Guide* in the NetSuite Help Center.

Parameters

- **id** {string} [required] - The custom scriptId specified on the SuiteSignOn record (see figure). NetSuite recommends 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.

The screenshot shows the SuiteSignOn record configuration form. The 'ID' field is highlighted with a red rectangle and contains the text '_wlf_sso_partner_portlet'. Other fields include 'NAME' (SSO Portlet), 'CONSUMER KEY' (sZmB7DXzHBZjwPlc), 'SHARED SECRET', 'CONFIRM SHARED SECRET', 'PARTNER ACCOUNT', and 'WEB SERVICES ACCESS' (set to 'Same As UI Role'). There are 'Save' and 'Cancel' buttons at the top.

If you do not create a custom scriptId, a system-generated ID will be generated for you after the SuiteSignOn record is saved. You can also use the system-generated ID as the **id** value.



Note: After the SuiteSignOn record is saved, both the scriptId and system-generated ID are prefixed with **customsso**.

To see a list of IDs for all SuiteSignOn records, go to the SuiteSignOn list page at (Setup > Integration > SuiteSignOn).

Returns

- URL, OAuth token, and any integration variables as a string

Throws

- SSS_SUITESIGNON_NOT_CONFIGURED
- SSS_INVALID_SUITESIGNON

Since

- Version 2009.2

Example 1

This sample shows how to use `nlapiOutboundSSO(id)` in a portlet script to create a reference to the SuiteSignOn record. After the portlet is added to the dashboard, the script is executed. The value of the `nlapiOutboundSSO` variable is passed to an `iframe`, which makes the http request to load the source.

```
// create a portlet object
function buildPortlet(portlet, column)
{
  // set a portlet title
  title = 'My Custom SSO Portlet!'
  portlet.setTitle(title)

  // pass the scriptId of the SuiteSignOn record
  var url = nlapiOutboundSSO('customsso_wlf_sso_partner_portlet');

  // create an iframe. It is the iframe that makes the http request to
  // load the content of the portlet.
  var content = '<iframe src="'+url+'" align="center" style="width: 100%; height: 600px; margin:0; border:0; padding:0"></iframe>';

  // render the content in your portlet
  portlet.setHtml( content );
}
```

Example 2

This sample shows how to use `nlapiOutboundSSO(id)` in a Suitelet to create a reference to the SuiteSignOn record. When the Suitelet opens and the content of the `iframe` is generated, the URL specified on the SignSignOn record will render.

```
function buildSuitelet(request, response)
{
  if ( request.getMethod() == 'GET' )
  {
    //create a form
    var form = nlapiCreateForm('SSO Suitelet');
    var label = form.addField('custpage_label', 'inlinehtml', 'SS01');
    label.setDefaultValue ( '<B>Check out my SSO Suitelet!!</B>' );

    var url = nlapiOutboundSSO('customsso_wlf_sso_partner_suitelet');
    var content = '<iframe src="'+url+'" align="center" style="width: 1000px; height: 800px; margin:0; border:0; padding:0"></iframe>';

    var iFrame = form.addField('custpage_sso', 'inlinehtml', 'SS02');
    iFrame.setDefaultValue ( content );
    iFrame.setLayoutType('outsidebelow', 'startcol');

    response.writePage( form );
  }
}
```

Example 3

This sample shows how to use `nlapiOutboundSSO(id)` in a user event script to integrate with an external application. At the point indicated by the user event script record (Before Load, Before Submit, or After Submit), the script gets the SuiteSignOn record that has this script defined as a connection point. The script returns the external application URL and any integration variables associated with this SuiteSignOn record and sends an http request to this URL. The external application can respond.

The most common usage of this type of script is to save a record in an external application when a record is saved in NetSuite.

```
function syncWithExternalApp(type)
{
    var url = nlapiOutboundSSO('customsso_my_external_app');
    nlapiRequestURL(url);
}
```

[Back to Communication APIs](#) | [Back to SuiteScript Functions](#)

Configuration APIs

NetSuite allows developers to programmatically obtain, and in some cases, change the values on certain account configuration pages. The internal IDs for SuiteScript-supported configuration pages are provided below. For the IDs that represent specific preferences on a configuration page, see the help topic [Preference Names and IDs](#) in the NetSuite Help Center.

All APIs listed below are in alphabetical order.

- [nlapiLoadConfiguration\(type\)](#)
- [nlapiSubmitConfiguration\(name\)](#)
- [nlobjConfiguration](#)

nlapiLoadConfiguration(type)

Use this API to load a NetSuite configuration page. The following configuration pages support SuiteScript: Company Information, General Preferences, User Preferences, Accounting Preferences, Accounting Periods, Tax Periods.

After a page is loaded, you can set configuration values using `nlobjConfiguration.setFieldValue(name, value)`.



Important: In most server-side scripts, addresses are accessed with the subrecord APIs (see the help topic [Scripting the Address Subrecord](#) for more information). Scripts that access addresses on the Company Information page are an exception to this rule. You must access address fields on the Company Information page the same way you access other fields. See [Example 1](#) for a code sample.

The `nlapiLoadConfiguration` function is available in scheduled scripts, user event scripts, and Suitelets. It consumes 10 usage units per call.

Parameters

- `type` - {string} [required] - The internal ID of the configuration page. Available IDs are:

- **companyinformation** - The internal ID for the Company Information page (Setup > Company > Company Information).
- **companypreferences** - The internal ID for the General Preferences page (Setup > Company > General Preferences).
- **userpreferences** - The internal ID for the Set Preferences page (Home > Set Preferences).
- **accountingpreferences** - The internal ID for the Accounting Preferences page (Setup > Accounting > Accounting Preferences).
- **accountingperiods** - The internal ID for the Accounting Periods page (Setup > Accounting > Manage Accounting Periods).
- **taxperiods** - The internal ID for the Tax Periods page (Setup > Accounting > Manage Tax Periods).
- **companyfeatures** - The internal ID for looking up which features are enabled in an account.

Returns

- `nlobjConfiguration` object

Since

- Version 2009.2

Example 1

This example loads the Company Information page and then accesses the shipping address. Note that you cannot use the subrecord APIs to access address fields on the Company Information page. Access these fields with `nlapiLoadConfiguration` in the same way you access other fields.

```
//load Netsuite configuration page
var companyInfo = nlapiLoadConfiguration('companyinformation');

//get field values
var ShipAddr1 = companyInfo.getFieldValue('shippingaddress1');
var shipCity = companyInfo.getFieldValue('shippingcity');
var shipState = companyInfo.getFieldValue('shippingstate');
var shipZip = companyInfo.getFieldValue('shippingzip');
var shipCountry = companyInfo.getFieldValue('shippingcountry');
```

Example 2

This example shows how to load the Company Information configuration page and then set the values for the Employer Identification Number (EIN) (**employerid**) field and the SSN or TIN (Social Security Number, Tax ID Number) (**taxid**) field.

```
//load the NetSuite configuration page
var companyInfo = nlapiLoadConfiguration('companyinformation');

//set field values
companyInfo.setFieldValue('employerid', '123456789');
companyInfo.setFieldValue('taxid', '1122334455');

//save changes to the configuration page
nlapiSubmitConfiguration(companyInfo);
```


[Back to Configuration APIs](#) | [Back to SuiteScript Functions](#)

nlapiSubmitConfiguration(name)

Use this API to submit changes to a configuration page that was loaded into the system using [nlapiLoadConfiguration\(type\)](#). The following configuration pages support SuiteScript: Company Information, General Preferences, Enable Features, Accounting Preferences, Accounting Periods, Tax Periods.

The **nlapiSubmitConfiguration** function is available in scheduled and Suitelet scripts only. It consumes 20 usage units per call.

Parameters

- **name** - {nlobjConfiguration} [required] - [nlobjConfiguration](#) object containing the data record

Returns

- void

Since

- Version 2009.2

Example

This example shows how to load the Company Information configuration page and then set the values for the Employer Identification Number (EIN) (**employerid**) field and the SSN or TIN (Social Security Number, Tax ID Number) (**taxid**) field.

```
// load the NetSuite configuration page
var companyInfo = nlapiLoadConfiguration( 'companyinformation' );

// set field values
companyInfo.setFieldValue( 'employerid', '123456789' );
companyInfo.setFieldValue( 'taxid', '1122334455' );

// save changes to the configuration page
nlapiSubmitConfiguration( companyInfo );
```

[Back to Configuration APIs](#) | [Back to SuiteScript Functions](#)

nlobjConfiguration

See [nlobjConfiguration](#) - defined in the section on [Standard Objects](#).

[Back to Configuration APIs](#) | [Back to SuiteScript Functions](#)

SuiteFlow APIs

Use these APIs to interact with the NetSuite SuiteFlow Manager.

All APIs listed below are in alphabetical order.

- `nlapiInitiateWorkflow(recordtype, id, workflowid, initialvalues)`
- `nlapiTriggerWorkflow(recordtype, id, workflowid, actionid, stateid)`

`nlapiInitiateWorkflow(recordtype, id, workflowid, initialvalues)`

Use this function to initiate a workflow on-demand. This function is the programmatic equivalent of the [Initiate Workflow Action](#) action in the SuiteFlow Manager. The function returns the workflow instance ID for the workflow-record combination. A user error is thrown if the record in the workflow is invalid or not supported for that workflow.

Usage metering allowed is 20 units. This API is supported in user event, scheduled, portlet, Suitelet, mass update, and workflow action scripts.

Parameters

- **recordtype** {string} [required] - The record type ID of the record type on which you are executing the workflow (for example, 'customer', 'salesorder', 'lead'). In the Workflow Manager, this is the record type that is specified in the Record Type field.
- **id** {int} [required] - The internal ID of the base record (for example 55 or 124).
- **workflowid** {int | string} [required] - The internal ID (int) or script ID (string) for the workflow definition. This is the ID that appears in the ID field on the [Workflow Definition Page](#).
- **initialvalues** {object} [optional] - Name/value pairs representing defaults used during workflow initialization.

Returns

- The internal ID (int) of the workflow instance used to track the workflow against the record.

Since

- Version 2010.1

[Back to SuiteFlow APIs](#) | [Back to SuiteScript Functions](#)

`nlapiInitiateWorkflowAsync(recordType, id, workflowId, initialValues)`

Use this function to asynchronously initiate a workflow. When you call `nlapiInitiateWorkflowAsync`, a job is created to initiate an instance of the specified workflow. The job is placed in the scheduling queue, and the workflow instance is initiated after the job reaches the top of the queue.



Note: `nlapiInitiateWorkflowAsync` does not successfully place a workflow job in queue if an identical instance of that workflow (with the same **recordType**, **id**, and **workflowId**) is currently executing or already in the scheduling queue.

The return value of `nlapiInitiateWorkflowAsync` is a string representing the workflow status. See [Returns](#) for additional information. An error is thrown if the record in the workflow is invalid or not supported for that workflow.

Usage metering allowed is 20 units. This API is supported in all server-side scripts.

Parameters

- **recordType** {string} [required] – The record type ID of the record type on which you are executing the workflow (for example, 'customer', 'salesorder', 'lead'). In the Workflow Manager, this is the record type that is specified in the Record Type field.
- **id** {int} [required] – The internal ID of the base record (for example 55 or 124).
- **workflowId** {int | string} [required] – The internal ID (int) or script ID (string) for the workflow definition. This is the ID that appears in the ID field on the [Workflow Definition Page](#).
- **initialValues** {object} [optional] – Name/value pairs representing defaults used during workflow initialization.

Returns

- A string value that indicates whether the workflow was successfully placed in the scheduling queue:
 - If the workflow job is successfully placed in queue, the return value is QUEUED.
 - If the workflow job is not successfully placed in queue, one of the following values is returned:
 - INQUEUE — Returned if the workflow is already in queue and waiting to run. If this status is returned, you must wait until the workflow job is finished before attempting to place another instance of the workflow in the queue.
 - INPROGRESS - Returned if the workflow is currently running.

Throws

Since

- Version 2014 Release 2

Example

nlapiTriggerWorkflow(recordtype, id, workflowid, actionid, stateid)

Use this API to trigger a workflow on a record. The actions and transitions of the workflow will be evaluated for the record based on the current state that it is in.

Usage metering allowed is 20 units. This API is supported in user event, scheduled, portlet, Suitelet, mass update, and workflow action scripts.

Beginning in Version 2015 Release 2, workflow action script ids are no longer guaranteed to be unique per workflow. Script ids may be the same for one or more actions and are identified by the parent workflow state. To support this new behavior, a parameter, stateid, has been added for nlapiTriggerWorkflow. The new parameter does not affect existing code and is not required in new code. However, if the stateid parameter is used, the actionid parameter is required. For example,

```
nlapiTriggerWorkflow('recordname', 123, 'workflow_id', 'workflowaction_id',
'state_id')
```

Parameters

- **recordtype** {string} [required] - The record type ID of the record type on which you are executing the workflow (for example, 'customer', 'salesorder', 'lead'). In the Workflow Manager this is the record type that is specified in the Record Type field.

- **id** {int} [required] - The internal ID of the base record (for example 55 or 124).
- **workflowid** {int | string } [required] - The internal ID (int) or script ID (string) for the workflow definition. This is the ID that appears in the ID field on the [Workflow Definition Page](#).
- **actionid** {string | int} [optional] - The internal ID of a button that appears on the record in the workflow. Using this parameter triggers the workflow as if the specified button were pressed.
- **workflowstateid** {string | int} [optional] - The internal ID (int) or script ID (string) of the state the action is in. This parameter can identify actions when a script id is used by more than one action in the same workflow. Requires use of the actionid parameter. If you choose not to use this parameter, NetSuite uses the action with the lowest internal ID.

Returns

- The internal ID (int) of the workflow instance used to track the workflow against the record.

Since

- Version 2010.1

[Back to SuiteFlow APIs](#) | [Back to SuiteScript Functions](#)

Portlet APIs

Use these APIs to work with NetSuite dashboard portlets.

All APIs listed below are in alphabetical order.

- [nlapiRefreshPortlet\(\)](#)
- [nlapiResizePortlet\(\)](#)

nlapiRefreshPortlet()

Causes a FORM type [nlobjPortlet](#) to immediately reload.

This API is available within a client SuiteScript associated with a custom FORM portlet, or from JavaScript event handlers attached to portlet elements. This API cannot be called directly from within a FORM portlet script.

Parameters

- None

Returns

- Void

Since

- Version 2011.1

Example

The following code adds a link that can be clicked to refresh a portlet on demand:

```
fld = portlet.addField('refrfield', 'inlinehtml', 'Refresh');
fld.setDefaultValue("<a onclick='nlapiRefreshPortlet()' href='#'>Refresh Now!</a>");
```

[Back to Portlet APIs](#) | [Back to SuiteScript Functions](#)

nlapiResizePortlet()

Causes a custom form portlet ([nlobjPortlet](#)) to be resized.

Custom form portlets are embedded in `<iframe>` elements (most other portlets are embedded in `<div>` elements). Browsers do not automatically resize `<iframe>` elements to fit their contents. If you change your custom form portlet content so that it no longer fits inside the portlet borders (whether the border is too small or too large), use the **nlapiResizePortlet** API to resize the portlet to fit your content.

This API is supported in client SuiteScripts associated with custom form portlets, or in JavaScript event handlers attached to portlet elements. This API cannot be called directly from within a FORM portlet script.

Parameters

- None

Returns

- Void

Since

- Version 2011.1

Example

The following example creates a small custom form portlet with a "Mutate!" link. When this link is clicked, a div element in the portlet is randomly resized and **nlapiResizePortlet** is called to adjust the portlet to match.

```
function demoSimpleFormPortlet(portlet, column)
{
    portlet.setTitle('nlapiResizePortlet demo');
    var txtField = portlet.addField('text', 'text', 'Random text field');
    txtField.setLayoutType('normal', 'startcol');

    var fld = portlet.addField('divfield', 'inlinehtml');
    fld.setDefaultValue("<div id='divfield_elem' style='border: 1px dotted red; height: 32px; width: 32px'></div>");

    fld = portlet.addField('growlink', 'inlinehtml');
    fld.setDefaultValue("<a onclick='mutate()' href='#'>Mutate!</a>");

    portlet.setScript('customscriptclienta');
}
```

```
function mutate()
{
    var div = document.getElementById('divfield_elem');
    var h = 32 + Math.floor(Math.random() * 128);
    div.style.height = h + 'px';

    nlapiResizePortlet();
}
```

[Back to Portlet APIs](#) | [Back to SuiteScript Functions](#)

SuiteAnalytics APIs

Use these APIs to work with NetSuite Analytics.

All APIs listed below are in alphabetical order.

- [nlapiCreateReportDefinition\(\)](#)
- [nlapiCreateReportForm\(title\)](#)
- [nlobjPivotColumn](#)
- [nlobjPivotRow](#)
- [nlobjPivotTable](#)
- [nlobjPivotTableHandle](#)
- [nlobjReportColumn](#)
- [nlobjReportColumnHierarchy](#)
- [nlobjReportDefinition](#)
- [nlobjReportForm](#)
- [nlobjReportRowHierarchy](#)

nlapiCreateReportDefinition()

Creates an instance of a report definition object. The report is built on this object using subsequent methods. The report definition can be used to create a form for rendering the pivot table report in a browser, or the pivot table APIs can be used to extract the values of the individual rows and columns of the pivot table.

Returns

- [nlobjReportDefinition](#)

Since

- Version 2012.2

Example

- See the code sample in [Building a Pivot Report Using SuiteScript](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjCreateReportForm(title)

Creates an `nlobjReportForm` object to render the report definition.

Parameters

- **title** {string} [required] - The title of the form.

Returns

- `nlobjReportForm`

Since

- Version 2012.2

Example

- See the code sample in [Building a Pivot Report Using SuiteScript](#).
[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjPivotColumn

See `nlobjPivotColumn` - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjPivotRow

See `nlobjPivotRow` - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjPivotTable

See `nlobjPivotTable` - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjPivotTableHandle

See `nlobjPivotTableHandle` - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjReportColumn

See `nlobjReportColumn` - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjReportColumnHierarchy

See [nlobjReportColumnHierarchy](#) - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjReportDefinition

See [nlobjReportDefinition](#) - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjReportForm

See [nlobjReportForm](#) - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

nlobjReportRowHierarchy

See [nlobjReportRowHierarchy](#) - defined in the section on [Standard Objects](#).

[Back to SuiteAnalytics APIs](#) | [Back to SuiteScript Functions](#)

User Credentials APIs

Use these APIs to change the NetSuite login credentials of the currently logged-in user. In NetSuite, a user's login credentials consists of a user's email address and a password.



Important: When building a custom UI outside of the standard NetSuite UI (such as building a custom mobile page using Suitelet or building E-Commerce pages using SSP), use these APIs to help users manage their credentials within the custom UI.

All APIs listed below are in alphabetical order.

- [nlapiGetLogin\(\)](#)
- [nlobjLogin](#)

nlapiGetLogin()

Returns the NetSuite login credentials of currently logged-in user.

This API is supported in user event, portlet, Suitelet, RESTlet, and SSP scripts. For information about the unit cost associated with this API, see the help topic [SuiteScript 1.0 API Governance](#).

Returns

- [nlobjLogin](#)

Since

- Version 2012.2

Example

This example shows how to get the credentials of the currently logged-in user.

```
//Get credentials of currently logged-in user
require(['N/runtime'],function(runtime){
  log.debug(JSON.stringify(runtime.getCurrentUser()));
})
```

[Back to User Credentials APIs](#) | [Back to SuiteScript Functions](#)

nlobjLogin

See [nlobjLogin](#) - defined in the section on [Standard Objects](#).

[Back to User Credentials APIs](#) | [Back to SuiteScript Functions](#)

Job Manager APIs

Use these APIs to send jobs to the internal job manager. Currently the job manager that is exposed to SuiteScript is the job manager that manages merging duplicate records.

When submitting a “merge duplicate record” job to NetSuite, SuiteScript supports all of the same functionality available through the UI. Using SuiteScript you can use the predefined duplicate detection rules, or you can define your own. Note that the merge duplicate API runs in server scripts, such as user event scripts, Suitelets, and RESTlets. You cannot write client scripts using this API.



Important: The merge duplicate functionality of non-entity records is not supported in SuiteScript.

After your records are merged/deleted, these records no longer appear as duplicates accessible through **nlapisearchDuplicate** or the UI (by going to Lists > Mass Update > Mass Duplicate Record Merge).

Finally, be aware that when you submit a merge duplicate job, the maximum number of records you can submit in your request is 200. Also be aware that then you call **nlobjJobManager.submit** to submit your job request, you are charged **100** governance units.

All APIs listed below are in alphabetical order.

- [nlapigetJobManager\(jobType\)](#)
- [nlobjJobManager](#)
- [nlobjDuplicateJobRequest](#)
- [nlobjFuture](#)

nlapigetJobManager(jobType)

Returns a job manager instance ([nlobjJobManager](#)). You then use the methods on **nlobjJobManager** to create and submit your merge duplicate records request. This API is supported in script types that run on the server. You cannot use this function in a client script.

This API costs no governance units.

Parameters

- **jobType** {string} [required] - Set to DUPLICATERECORDS.

Returns

- [nlobjJobManager](#)

Since

- Version 2013.1

Example - Using the Job Manager APIs to Merge Duplicate Records

```
function mergeLeads() {

// Get all duplicate lead records that have the same email address
var fldMap = new Array();
fldMap['email'] = 'user@testing123.com'
var duplicateRecords = nlapiSearchDuplicate( 'lead', fldMap );
var arrID = new Array();
var record;

for ( var i = 0; i < duplicateRecords.length; i++ )
{
    var duplicateRecord = duplicateRecords[ i ];
    arrID[i] = duplicateRecord.getId( );
}

// Get a job manager instance.
var manager = nlapiGetJobManager( 'DUPLICATERECORDS' );

// Create the merge job object.
var mergeJobRequest = manager.createJobRequest();

// Set the entity type.
mergeJobRequest.setEntityType(mergeJobRequest.ENTITY_LEAD);

// Set the master. The master can be manually indicated or found by criteria.
mergeJobRequest.setMasterSelectionMode(mergeJobRequest.MASTERSELECTIONMODE_CREATED_
    EARLIEST);

// Set duplicate records. Pass in parameter as an array of duplicate record IDs
mergeJobRequest.setRecords(arrID);

// Set the merge operation type.
mergeJobRequest.setOperation(mergeJobRequest.OPERATION_MERGE);

// Submit a job to process asynchronously. Submitting the job does not execute the job.
// Submitting the job places the job in the queue.
jobId = manager.submit(mergeJobRequest);

// Check the job status
var future = manager.getFuture(jobId);
```

```
// See if job has completed.  
future.isDone();  
  
// See if job has been cancelled. Note, for merge duplicate records, this method will always return false  
future.isCancelled();  
  
}
```

For more details about the methods used in this example, see [nlobjJobManager](#), [nlobjDuplicateJobRequest](#), and [nlobjFuture](#).

Back to [Job Manager APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjJobManager

See [nlobjJobManager](#) - defined in the section on [Standard Objects](#).

Back to [Job Manager APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjDuplicateJobRequest

See [nlobjDuplicateJobRequest](#) - defined in the section on [Standard Objects](#).

Back to [Job Manager APIs](#) | **Back to** [SuiteScript Functions](#)

nlobjFuture

See [nlobjFuture](#) - defined in the section on [Standard Objects](#).

Back to [Job Manager APIs](#) | **Back to** [SuiteScript Functions](#)

SuiteScript Objects

SuiteScript Objects Overview

SuiteScript objects are classified into the following two categories. Click the links below to see which objects are assigned to each category. From there you can also access API documentation for each method on the object.

- [Standard Objects](#)
- [UI Objects](#)

Standard Objects

The objects in this list are **standard** objects. Unlike [UI Objects](#), they are not used to build NetSuite UI components such as buttons, forms, fields, sublists, etc. Standard objects are used more for manipulating backend data and to handle form GET and POST processing.

Each standard object has methods that can be performed against it when it is returned in the script. The following is a list of all **standard** NetSuite objects.

- [nlobjConfiguration](#)
- [nlobjContext](#)
- [nlobjCredentialBuilder\(string, domainString\)](#)
- [nlobjCSVImport](#)
- [nlobjDuplicateJobRequest](#)
- [nlobjEmailMerger](#)
- [nlobjError](#)
- [nlobjFile](#)
- [nlobjFuture](#)
- [nlobjJobManager](#)
- [nlobjLogin](#)
- [nlobjMergeResult](#)
- [nlobjPivotColumn](#)
- [nlobjPivotRow](#)
- [nlobjPivotTable](#)
- [nlobjPivotTableHandle](#)
- [nlobjRecord](#)
- [nlobjReportColumn](#)
- [nlobjReportColumnHierarchy](#)
- [nlobjReportDefinition](#)
- [nlobjReportForm](#)
- [nlobjReportRowHierarchy](#)

- [nlobjRequest](#)
- [nlobjResponse](#)
- [nlobjSearch](#)
- [nlobjSearchColumn\(name, join, summary\)](#)
- [nlobjSearchFilter](#)
- [nlobjSearchResult](#)
- [nlobjSearchResultSet](#)
- [nlobjSelectOption](#)
- [nlobjSubrecord](#)

nlobjConfiguration

Primary object used to encapsulate a NetSuite configuration/setup page. Note that [nlapiLoadConfiguration\(type\)](#) returns a reference to this object. After the **nlobjConfiguration** object has been modified, changes can be submitted to the database using [nlapiSubmitConfiguration\(name\)](#).

For a list of configuration pages that support SuiteScript, see the help topic [Preference Names and IDs](#) in the NetSuite Help Center.

nlobjConfiguration Methods

- [getAllFields\(\)](#)
- [getField\(fldnam\)](#)
- [getFieldText\(name\)](#)
- [getFieldTexts\(name\)](#)
- [getFieldValue\(name\)](#)
- [getFieldValues\(name\)](#)
- [getType\(\)](#)
- [setFieldText\(name, text\)](#)
- [setFieldTexts\(name, text\)](#)
- [setFieldValue\(name, value\)](#)
- [setFieldValues\(name, value\)](#)

getAllFields()

Use this method to return a normal keyed array of all the field names on a configuration page.

Returns

- `String[]` of field names

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getField(fldnam)

Use the method to return field metadata for a field

Parameters

- **fldnam** {string} [required] - The internal ID of the field

Returns

- The [nlobjField](#) object

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldText(name)

Use this method to return the UI display value for a select field. This API is supported in select fields only.

Parameters

- **name** {string} [required] - The internal ID of the field

Returns

- String - The UI display value corresponding to the current selection for a select field. Returns *null* if field does not exist on the configuration page or if the field is restricted.

Since

- Version 2009.2

Example

This sample shows how to use **getFieldText(name)** to return the UI display value for the First Day of Week configuration preference. In this account, First Day of Week has been set to **Sunday**. This is the value that will be returned.

```
var configpage = nlapiLoadConfiguration('companypreferences');
var valtext = configpage.getFieldText('firstdayofweek'); // returns Sunday
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldTexts(name)

Use this method to return the UI display values for a multiselect field

Parameters

- **name** {string} [required] - The name of the multiselect field whose field display values are being returned

Returns

- Returns the selected text values of a multiselect field as an Array

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValue(name)

Use this method to return the internal ID value of a field

Parameters

- **name** {string} [required] - The internal ID of the field

Returns

- The internal ID (string) value for the field

Since

- Version 2009.2

Example

```
// load an Accounting Periods configuration page
var configpage = nlapiLoadConfiguration('accountingpreferences');

// get value of the Cash Basis field. The value F will be returned since this is a
//checkbox field that is not selected.
var value = configpage.getFieldValue('cashbasis');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValues(name)

Returns a **read-only** array of multi-select field values. This API is supported on multi-select fields only.

Parameters

- **name** {string} [required]- The internal ID of the field

Returns

- String[] of field IDs. Returns **null** if field is not on the configuration page.

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getType()

Use this method to return the internal ID of a configuration page, for example, **accountingpreferences** or **taxperiods**.

Returns

- The internal ID of the configuration page as a string

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldText(name, text)

Use this method to set the value of a select field using its corresponding display value. This API is supported on select fields only.

Parameters

- **name** {string} [required] - The internal ID of the field being set
- **text** {string} [required] - The field display name as it appears in the UI

Returns

- void

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldTexts(name, text)

Use this method to set the values (via the UI display values) of a multi-select field. This API is supported on multi-select fields only.

Parameters

- **name** {string} [required] - The internal ID of the field being set

- **texts** {string[]} [required] - Array of field display values

Returns

- void

Since

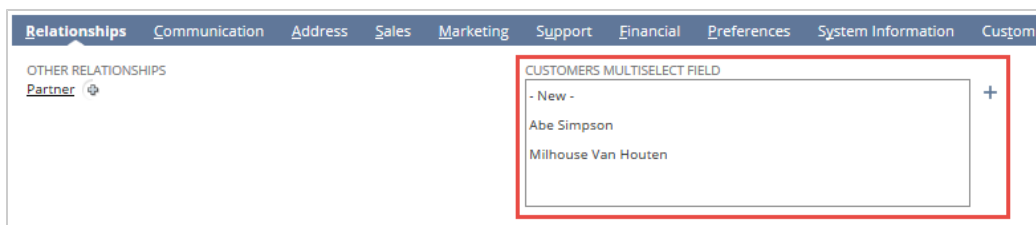
- Version 2009.2

Example

```
var values = new Array(); // create an array of customers who are currently in NetSuite
values[0] = 'Abe Lincoln'; // add the first customer
values[1] = 'Abe Simpson'; // add the second customer
var record = nlapiLoadRecord('salesorder', 447); // load the sales order

// set the field display values for the custom multiselect field
// called Customers Multiselect Field
record.setFieldTexts('custbody16', values);

// submit the record
var submit = nlapiSubmitRecord(record, true);
```



[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValue(name, value)

Use this method to set the value of a field

Parameters

- **name** {string} [required] - The internal ID of the field being set
- **value** {string} [required] - The value the field is being set to

Returns

- void

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValues(name, value)

Use this method to set the value of a multi-select field. This API is supported on multi-select fields only.

Parameters

- **name** {string} [required] - The internal ID of the field being set
- **value** {string[]} [required]- The value the field is being set to

Returns

- void

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjContext

Encapsulates user information as well as script execution context at runtime. Note that the [nlobjGetContext\(\)](#) function returns a reference to this object.

nlobjContext Methods

- [getBundleIds\(\)](#)
- [getColorPreferences\(\)](#) (Deprecated as of Version 2014 Release 2)
- [getCompany\(\)](#)
- [getDepartment\(\)](#)
- [getDeploymentId\(\)](#)
- [getEmail\(\)](#)
- [getEnvironment\(\)](#)
- [getExecutionContext\(\)](#)
- [getFeature\(name\)](#)
- [getLocation\(\)](#)
- [getLogLevel\(\)](#)
- [getName\(\)](#)
- [getPercentComplete\(\)](#)
- [getPermission\(name\)](#)
- [getPreference\(name\)](#)
- [getProcessorCount\(\)](#)
- [getQueueCount\(\)](#)
- [getRemainingUsage\(\)](#)
- [getRole\(\)](#)

- `getRoleCenter()`
- `getRoleId()`
- `getScriptId()`
- `getSessionObject(name)`
- `getSetting(type, name)`
- `getSubsidiary()`
- `getUser()`
- `getVersion()`
- `setPercentComplete(pct)`
- `setSessionObject(name, value)`
- `setSetting(type, name, value)`

getBundleIds()

Returns the bundle IDs for the current script.


Returns

Returns an Array of bundle IDs for the bundles that include the currently executing script.

Since

Version 2016 Release 1

getColorPreferences()

 **Note:** This method is deprecated as of Version 2014 Release 2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCompany()

Returns the currently logged in user's account ID

Returns

- The string value of user's account ID, for example NL555ABC

Since

- Version 2007.0

Example

```
var context = nlapiGetContext();
var userAccountId = context.getCompany();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getDepartment()

Returns the internal ID of the currently logged in user's department

Returns

- The logged in user's department ID as an integer

Since

- Version 2007.0

Example

```
var context = nlapiGetContext();
var userDeptId = context.getDepartment();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getDeploymentId()

Returns the deploymentId for the current script deployment (ie., the currently executing script)

Returns

- The deploymentId as a string

Since

- Version 2009.1


Example

- In the API documentation for [nlapiScheduleScript\(scriptId, deployId, params\)](#), see [Example 1 - Rescheduling a Script](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getEmail()

Returns the currently logged in user's e-mail address. The **email** field on the user's employee record must contain an email address.

 **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.

Returns

- An email address as a string

Since

- Version 2007.0

Example

```
var context = nlapiGetContext();
var userEmail = context.getEmail();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getEnvironment()

Returns the environment in which the current script is being executed. Valid values are SANDBOX | PRODUCTION | BETA | INTERNAL.

Returns

- The name of the environment as a string

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getExecutionContext()

Returns context information about what triggered the current script. Possible return values are:

- **userinterface** - Client SuiteScript or user event triggers invoked from the UI
- **webservices** - User event triggers invoked from webservice calls
- **csvimport** - User event triggers invoked during CSV imports
- **portlet** - Portlet script or user event triggers invoked via portlet scripts
- **scheduled** - Scheduled script or user event triggers invoked via scheduled scripts
- **suitelet** - Suitelet or user event triggers invoked via suitelets
- **custommassupdate** - Mass update script triggers invoked via custom Mass Update scripts
- **workflow** - Workflow action script triggers invoked via Workflow Action scripts
- **webapplication** - Suitelet or user event triggers invoked via SSP application files
- **webstore** - User event triggers invoked from the web store (for example to determine if sales orders or customers were created in the web store).
- **userevent** - This context type represents cases in which records are generated in the backend (as opposed to being generated by the UI). For example, the 'userevent' context distinguishes the case wherein a Bill Payment is submitted as part of a non-record page. Whereas the 'userinterface' context identifies when a single Bill Payment record is submitted from the UI.

Returns

- The execution context as a string

Since

- Version 2007.0

Example

This is a beforeLoad user event script deployed on the Case record. When `getExecutionContext` returns `userinterface` and `type` is 'edit' or 'view', a tab is added to the Case record.

```
function caseBeforeLoad(type, form)
{
    var currentContext = nlapiGetContext();
    if( (currentContext.getExecutionContext() == 'userinterface') && (type == 'edit' | type == 'view'))
    {
        var SampleTab = form.addTab('custpage_sample_tab', 'SampleTab123');
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFeature(name)

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 (Setup > Company > Enable Features).

Parameters

- **name** {string} [required] - The internal ID of the feature. For a list of feature IDs, see the help topic [Feature Names and IDs](#) in the NetSuite Help Center.

Returns

- Returns true if a feature is enabled in the current account

Since

- Version 2009.2

Example

This sample shows how to determine whether the Advanced Billing feature is enabled in your account.

```
var context = nlapiGetContext();
context.getFeature('ADVBILLING');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLocation()

Returns the internal ID of the currently logged in user's location

Returns

- The logged in user's location ID as an integer

Since

- Version 2007.0

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLogLevel()

Returns the script logging level for the current script execution. This method is not supported on client scripts.

Returns

- The string value of the script log level. Possible values are DEBUG, AUDIT, ERROR, EMERGENCY

Since

- Version 2008.2

See also

- [nlapiLogExecution\(type, title, details\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getName()

Returns the currently logged in user's name



Note: In a shopping context where the shopper is recognized but not logged in, this method can be used to return the shopper's name, instead of getting it from the customer record.

Returns

- The logged in user's name as a string

Since

- Version 2007.0

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getPercentComplete()

Return the % complete specified for the current scheduled script execution. The return value will appear in the **%Complete** column in the Scheduled Script Status page. Note that this method can only be called from scheduled scripts.

Returns

- The integer value of the percent complete field

Since

- Version 2009.1

Example

The following script is a scheduled script that performs a customer search. Use the **setPercentComplete** and **getPercentComplete** methods to define percentage complete values and then get the values. When **getPercentComplete** is called, the value appears in the **%Complete** column in the Scheduled Script Status page. Access this page by going to Customization > Scripting > Script Deployments.

```
function customerSearch(type)
{
    var ctx = nlapiGetContext(); // instantiate the nlobjContext object
    var searchresults = nlapiSearchRecord('customer', 21); // execute a specific saved search
    ctx.setPercentComplete(0.00); // set the percent complete parameter to 0.00

    for ( i = 0; i < searchresults.length; i++ ) // loop through the search results
    {

        // get the internal ID of each returned record, otherwise you cannot update the results
        var recid = searchresults[i].getValue('internalid');

        var record = nlapiLoadRecord('customer', recid); // load each record from the search
        record.setFieldText('salesrep', 'John Doe'); // set a field display value for Sales Rep
        var id = nlapiSubmitRecord(record, true); // submit the record
        ctx.setPercentComplete( (100* i)/ searchresults.length ); // calculate the results

        // displays the percentage complete in the %Complete column on
        // the Scheduled Script Status page
        ctx.getPercentComplete(); // displays percentage complete
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getPermission(name)

Use this method to get a user's permission level for a specific permission. For information on working with NetSuite permissions, see the topic *Understanding NetSuite Permissions* in the NetSuite Help Center.

Parameters

- **name** {string} [required] - The internal ID of a permission. For a list of permission IDs, see the help topic [Permission Names and IDs](#) in the *SuiteScript Reference Guide*.

Returns

- The integer value of user's permission level for a specific permission. Values **4** through **0** can be returned:
 - **4** (FULL)
 - **3** (EDIT)
 - **2** (CREATE)
 - **1** (VIEW)
 - **0** (NONE)

Since

- Version 2009.2

Example

This sample shows how to determine a user's permission level for the Set Up Accounting permission.

```
var context = nlapiGetContext();
context.getPermission('ADMI_ACCOUNTING');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getPreference(name)

Use this method to get the value of a NetSuite preference. Currently only **General Preferences** and **Accounting Preferences** are exposed in SuiteScript. (You can 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, you must load each preference page using [nlapiLoadConfiguration\(type\)](#), where **name** is either 'companypreferences' (for the General Preferences page) or 'accountingpreferences' (for the Accounting Preferences page). The [nlapiLoadConfiguration](#) API returns an [nlobjRecord](#) object, which lets you change preference values using the [setFieldValue](#) method. For additional details, see [nlapiLoadConfiguration](#).

Note: The permission level will be 4 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.

Parameters

- **name** {string} [required] - The internal ID of the preference. For a list of preference IDs, see the help topic [Preference Names and IDs](#) in the NetSuite Help Center.

Returns

- The value of a system or script preference for the current user. The value can be **T** or **F** if the preference is a NetSuite check box field. The value can also be a string if the preference is a NetSuite dropdown field.

Since

- Version 2009.2

Example

This sample shows how to get the value of a NetSuite preference called Email Employee on Approvals.

```
var context = nlapiGetContext();
context.getPreference('emailemployeeonapproval');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getProcessorCount()

Returns 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.

For scheduled script deployments that continue to use queues, use `nlobjContext.getQueueCount()`. With the introduction of SuiteCloud Processors, 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](#)).

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](#).



Note: The `nlobjContext.getProcessorCount()` method returns 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](#).

For more information on scheduled scripts, see the help topic [Scheduled Scripts](#).

Returns

- The number of queues

Since

- Version 2013.1

Example

```
var queues = nlobjContext.getQueueCount();

if (queues == 5){

    // optimize for 5 queues

} else {

    // optimize for 1 queue

}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getQueueCount()

Returns the number of queues available to the currently logged in account.

[SuiteCloud Processors](#) is the current system used to execute (process) scheduled 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.

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](#)).

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 returned by `nlobjContext.getQueueCount()` is unchanged. This method returns the number of queues available to an account. It is not impacted by changes to deployments.

For more information on scheduled scripts, see the help topic [Scheduled Scripts](#).

Returns

- The number of queues

Since

- Version 2013.1

Example

```
var queues = nlobjContext.getQueueCount();

if (queues == 5){

    // optimize for 5 queues

} else {

    // optimize for 1 queue

}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRemainingUsage()

Returns the remaining amount of unit usage for the current script

Returns

- The integer value of the remaining unit count

Since

- Version 2007.0

Example

```
var context = nlapiGetContext();
```

```
var usageRemaining = context.getRemainingUsage();
```

See also

- [SuiteScript Governance](#) in the NetSuite Help Center
- [nlapiGetContext\(\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRole()

Returns the internal ID of the currently logged in user's role

Returns

- The logged in user's role ID as a string

Since

- Version 2007.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRoleCenter()

Returns the internal ID of the currently logged in user's center type (role center)

Returns

- The string value of the logged in user's center - for example, SALES, ACCOUNTING, CLASSIC. Note that the string value of a custom center can also be returned.

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRoleId()

Returns the custom scriptId of the role (as opposed to the internal numerical ID).

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 **getRoleId** to return the unique/custom scriptId assigned to the role.

Returns

- Custom scriptId of a role as a string.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getScriptId()

Returns the scriptId for the currently executing script

Returns

- The scriptId as a string

Since

- Version 2009.1

Example

- In the API documentation for [nlapiScheduleScript\(scriptId, deployId, params\)](#), see [Example 1 - Rescheduling a Script](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSessionObject(name)

Use this method to get the value of a user-defined session object for the current user.

Parameters

- **name** {string} [required] - The key used to store the session object

Returns

- Returns the string value of a user-defined session object for the current user

Since

- Version 2009.2

Example

This example shows how to get the value of the current user's session, and then create a new "Contact" session for the user to gather information about the user's scope, budget, and business problem.

```
function displayContact(request, response)
{
    var ctx = nlapiGetContext();
    var step = ctx.getSessionObject('stage');
```

```

if( step == null || step == "" )
{
    step = "create";
    ctx.setSessionObject('stage', 'Contact');
}
if(step == "create");
{
    ctx.setSessionObject('scope', request.getParameter('scope') );
    ctx.setSessionObject('approved', request.getParameter('budget') );
    ctx.setSessionObject('problem', request.getParameter('businessproblem') );
}
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSetting(type, name)

Use this API to get a system or script setting. Note that if you want to get session, feature, or permission settings directly, you can also use these **nlobjContext** methods:

- [getSessionObject\(name\)](#)
- [getFeature\(name\)](#)
- [getPermission\(name\)](#)

Parameters

- **type** {string} [required] - The type of script/system setting. Possible values include:
 - **SESSION** - session variable (volatile setting defined per session). Supported in server scripts only.



Important: The SESSION type value is not supported in Client SuiteScript.

- **FEATURE** - returns T (enabled) or F (disabled) depending on whether a feature is enabled. Supported in client and server SuiteScript.
In the NetSuite Help Center, see the help topic [Feature Names and IDs](#) for feature names and internal IDs.
- **PERMISSION** - returns permission level: 0 (none), 1 (view), 2 (create), 3 (edit), 4 (full). Supported in client and server SuiteScript.
In the NetSuite Help Center, see the help topic [Permission Names and IDs](#) for permission names and internal IDs.
- **SCRIPT** - script parameter (defined per script). Supported in client and server SuiteScript. If you do not know what script parameters are in NetSuite, see the help topic [Creating Script Parameters Overview](#).

- **name** {string} [required] - The name of the script/system setting



Important: You must use the nlobjContext. **getSetting** method to reference script parameters. For example, to obtain the value of a script parameter called **custscript_case_field**, you use the following code:

```
nlapigetContext().getSetting('SCRIPT', 'custscript_case_field')
```

If you do not know what script parameters are in NetSuite, see the help topic [Creating Script Parameters Overview](#).

Returns

- If **type** is specified as SCRIPT, SESSION, or FEATURE, a string value is returned. If **type** is specified as PERMISSION, an integer value is returned.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSubsidiary()

Returns the internal ID of the currently logged in user's subsidiary

Returns

- The logged in user's subsidiary ID as an integer

Since

- Version 2007.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getUser()

Returns the currently logged in user's internal ID

Returns

- The logged in user's ID as a string

Since

- Version 2007.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getVersion()

Returns the version of NetSuite that the method is called in. For example, if **getVersion** is executed in an account running NetSuite 2010.2, the value returned is **2010.2**. If **getVersion** is executed in an account running NetSuite 2010.1, the value returned is **2010.1**.

This method may be helpful to those installing SuiteBundles in other NetSuite accounts, and wish to know the version number before installing the bundle.

Returns

- The NetSuite account version as a number - for example: 2010.2

Since

- Version 2010.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setPercentComplete(pct)

Sets the percent complete for the currently executing scheduled script. Note that this method can only be called from scheduled scripts.

Parameters

- **pct** {float} [required] - The percentage of records completed

Returns

- void

Since

- Version 2009.1

Example

The following script is a scheduled script that performs a customer search. Use the **setPercentComplete** and **getPercentComplete** methods to define percentage complete values and then get the values. When **getPercentComplete** is called, the value appears in the **%Complete** column in the Scheduled Script Status page. Access this page by going to Customization > Scripting > Script Deployments. See the help topic [Use the Status Page or Status Links](#) for more information about this page.

```
function customerSearch(type)
{
    var ctx = nlapiGetContext();    // instantiate the nlobjContext object
    var searchresults = nlapiSearchRecord('customer', 21);    // execute a specific saved search
    ctx.setPercentComplete(0.00);    // set the percent complete parameter to 0.00

    for ( i = 0; i < searchresults.length; i++ ) // loop through the search results
    {

        // get the internal ID of each returned record, otherwise you cannot update the results
        var recid = searchresults[i].getValue('internalid');

        var record = nlapiLoadRecord('customer', recid);    // load each record from the search
        record.setFieldText('salesrep', 'John Doe');    // set a field display value for Sales Rep
        var id = nlapiSubmitRecord(record, true);    // submit the record
        ctx.setPercentComplete( (100* i)/ searchresults.length );    // calculate the results

        // displays the percentage complete in the %Complete column on
        // the Scheduled Script Status page
        ctx.getPercentComplete();    // displays percentage complete
    }
}
```


[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSessionObject(name, value)

Use this method to add or set the value of a user-defined session object for the current user. This value is valid during the current user's login.

This call allows the user to temporarily save something to the session before persisting it in a custom record.

Parameters

- **name** {string} [required] - The key used to store the session object
- **value** {string} [required] - The value to associate with this key in the user's session

Returns

- void

Since

- Version 2009.2

Example

This example shows how to get the value of the current user's session, and then create a new "Contact" session for the user to gather information about the user's scope, budget, and business problem.

```
function displayContact(request, response)
{
    var ctx = nlapiGetContext();
    var step = ctx.getSessionObject('stage');

    if( step == null || step == "" )
    {
        step = "create";
        ctx.setSessionObject('stage', 'Contact');
    }
    if(step == "create");
    {
        ctx.setSessionObject('scope', request.getParameter('scope') );
        ctx.setSessionObject('approved', request.getParameter('budget') );
        ctx.setSessionObject('problem', request.getParameter('businessproblem') );
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSetting(type, name, value)

Sets the value of a script or user-defined setting. Only available in server scripts.

- **type** {string} [required] - The type of script/system setting

- **SESSION** - session variable (volatile setting defined per session)
- **name** {string} [required]- The name of the script/system setting
- **value** {string} [required]- The new value for the script/system setting

Returns

- void



Important: You can also use the `nlobjContext.getSessionObject(name)` method to set session variable directly.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjCredentialBuilder(string, domainString)

The `nlobjCredentialBuilder` object encapsulates a request string that can be passed to `nlapirequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)`. Six methods are included that perform various string transformations: three hash methods for SHA-1, SHA-256, and MD5 hashing, two encoding methods for Base64 and UTF8 encoding, a character replacement method, and an appending method.



Important: If the `nlobjCredentialBuilder` object is passed to `nlapirequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)` as the `url` argument, it must be passed in its original state (pre-encryption and pre-encoding). Otherwise, `nlapirequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)` is unable to validate the URL.

The `nlobjCredentialBuilder` object is defined with the `new` keyword.

```
var builder = new nlobjCredentialBuilder("rawtext{GUID}Hash:", 'www.netsuite.com');
```

For a script sample that demonstrates how to use `nlobjCredentialBuilder`, see `nlapirequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)`, [Example 3](#).

Supported Script Types

- User Event
- Scheduled Script
- Portlet
- Suitelet

Parameters

- **string** {string} [required] – request string; can include an embedded GUID (globally unique string).
- **domainString** {string} [required] – URL's host name. Host name must exactly match the host name in your URL. For example, if your URL is `https://payment.ns.com/process.money?passwd={GUID}`, the host name passed in must be `'payment.ns.com'`.

nlobjCredentialBuilder Methods

- [append\(nlobjCredentialBuilder\)](#)

- `base64()`
- `md5()`
- `replace(string1, string2)`
- `sha1()`
- `sha256()`
- `utf8()`

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

append(nlobjCredentialBuilder)

Appends an **nlobjCredentialBuilder** object to another **nlobjCredentialBuilder** object.

Pass in **nlobjCredentialBuilder** object as input for this method. For example:

```
var builder = new nlobjCredentialBuilder("rawtext{GUID}Hash:", "www.netsuite.com");
builder = builder.md5();
var builder2 = new nlobjCredentialBuilder("rawtext{GUID}Hash:", "www.netsuite.com");
builder.append(builder2);
```

Parameter

- **object** {object} [required] — **nlobjCredentialBuilder** object to be appended.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

base64()

Encodes an **nlobjCredentialBuilder** object per the base64 scheme.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

Example

```
//builder contains content that is SHA-1 encrypted and then Base64 encoded
```

```
builder = builder.sha1().base64();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

md5()

Hashes an **nlobjCredentialBuilder** object with the MD5 hash function.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2015 Release 1

Example

```
//builder contains content that is MD5 hashed and then UTF-8 encoded
builder = builder.md5().utf8();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

replace(string1, string2)

Replaces all instances of **string1** with **string2**.

Parameters

- **string1** {string} [required] — string to be replaced
- **string2** {string} [required] — string to be replaced with

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

Example

```
//replace all instaces of "#" with "-" within builder
builder = builder.replace('#', '-');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

sha1()

Hashes an **nlobjCredentialBuilder** object with the SHA-1 hash function.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

Example

```
//builder contains content that is SHA-1 hashed and then Base64 encoded
builder = builder.sha1().base64();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

sha256()

Hashes an **nlobjCredentialBuilder** object with the SHA-256 hash function.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

Example

```
//builder contains content that is SHA-256 hashed and then UTF-8 encoded
builder = builder.sha256().utf8();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

utf8()

Encodes an **nlobjCredentialBuilder** object per the UTF-8 scheme.

Returns

- An **nlobjCredentialBuilder** object.

Since

- Version 2013 Release 2

Example

```
//builder contains content that is SHA-256 hashed and then UTF-8 encoded
builder = builder.sha256().utf8();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjCSVImport

Primary object used to encapsulate a CSV import job. This object is passed as a parameter to [nlapiSubmitCSVImport\(nlobjCSVImport\)](#), which is used to asynchronously import record data into NetSuite.

Note: CSV Imports performed within scripts are subject to the existing application limit of 25,000 records.

Use [nlapiCreateCSVImport\(\)](#) to return an **nlobjCSVImport** object. You can then use the object's methods to populate it with the desired information.

nlobjCSVImport Methods

- [setLinkedFile\(sublist, file\)](#)
- [setMapping\(savedImport\)](#)
- [setOption\(option, value\)](#)
- [setPrimaryFile\(file\)](#)
- [setQueue\(string\)](#)

Warning: You should execute [setMapping\(savedImport\)](#) before any of the other methods. If you try to first execute [setPrimaryFile\(file\)](#), an error is returned.

setLinkedFile(sublist, file)

Sets the data to be imported in a linked file for a multi-file import job, by referencing a file in the file cabinet using [nlapiLoadFile\(id\)](#), or by inputting CSV data as raw string.

If an import job requires multiple linked files, this method can be executed multiple times, one time for each linked file.

Parameters

- **sublist** {string} [required] — The internal ID of the record sublist for which data is being imported.
- **file** {string} [required] - Can be one of the following:
 - An **nlobjFile** object, encapsulating a CSV file, that contains the data to be imported. The CSV file must be uploaded to the file cabinet before it can be used in this context. The **nlobjFile** object is loaded with [nlapiLoadFile\(id\)](#). To load the **nlobjFile** object, pass the internal ID of the specific CSV file to be loaded, as shown below. The internal ID of the CSV file is listed in the file cabinet, under the Internal ID column.

```
setLinkedFile("item", nlapiLoadFile(74));
```

- Raw string of the data to be imported.

Returns

- void

Throws

- **SSS_INVALID_CSV_CONTENT** — Thrown when an invalid value is passed as the file argument.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMapping(savedImport)

Sets the name of the saved import map to be used for an import, by referencing the internal ID or script ID of the import map.

Parameters

- **savedImport** {string} [required] - The internal ID or script ID of the saved mapping to use for the import job. The internal ID is system-defined and is displayed in the ID column at Setup > Import/Export > Saved CSV Imports. The script ID can be defined in the Import Assistant and is also displayed on this page.

Returns

- void

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setOption(option, value)

Sets the name of the import job to be shown on the status page for CSV imports.

Parameters

- **option** {string} [required] - The name of the option, in this case, jobName.
- **value** {string} [required] - The value for the jobName option, meaning the text to be displayed in the Job Name column at Setup > Import/Export > View CSV Import Status. The default job name format is: <import type> - <csv file name> - <email address of logged-in user>.

Returns

- void

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setPrimaryFile(file)

Sets the data to be imported in the primary file for an import job, by referencing a file in the file cabinet using **nlapiloadFile**, or by inputting CSV data as raw string.

Parameters

- **file** {string} [required] - Can be one of the following:
 - The internal ID, as shown in the file cabinet, of the CSV file containing data to be imported, referenced by **nlapiloadFile**. For example:
`setPrimaryFile(nlapiloadFile(73))`
 - Raw string of the data to be imported.

Returns

- void

Throws

- SSS_INVALID_CSV_CONTENT — Thrown when an invalid value is passed as the file argument.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setQueue(string)

Overrides the CSV import queue preference. The stored queue preference is not altered; **setQueue** must be called each time an override is needed.



Note: This method is intended for users with a SuiteCloud Plus license.

Parameters

- **string** {string} [required] — The new queue number. Valid values range from '1' to '5', depending upon the SuiteCloud License.

Returns

- void

Throws

- SSS_INVALID_CSV_QUEUE — Thrown for all invalid values passed as the string argument.

Since

- Version 2014 Release 1

Example

```
var import1 = nlapiCreateCSVImport();
import1.setMapping('CUSTIMPORTImport1');
import1.setPrimaryFile(nlapiLoadFile(252)); //internal id of first csv file
nlapiSubmitCSVImport(import1); // run in queue defined on CUSTIMPORTImport1

var import2 = nlapiCreateCSVImport();
import2.setMapping('CUSTIMPORTImport1');
import2.setPrimaryFile(nlapiLoadFile(253)); //internal id of first csv file
import2.setQueue('2'); // run in queue 2
nlapiSubmitCSVImport(import2);

var import3 = nlapiCreateCSVImport();
import3.setMapping('CUSTIMPORTImport1');
import3.setPrimaryFile(253); // SSS_INVALID_CSV_CONTENT expected
import3.setQueue(6); // SSS_INVALID_CSV_QUEUE expected
nlapiSubmitCSVImport(import3);
```


[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjDuplicateJobRequest

Primary object used to encapsulate all the properties of a merge duplicate record job request. Note that `nlobjJobManager.createJobRequest()` returns a reference to this object.

Use the methods in `nlobjDuplicateJobRequest` to define the criteria of your merge duplicate request.

For an end-to-end example that shows how the job manager APIs work together, see [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

 **Note:** When submitting a merge duplicate job, the maximum number of records you can submit is 200.

nlobjDuplicateJobRequest Methods

- `setEntityType(entityType)`
- `setMasterId(masterID)`
- `setMasterSelectionMode(mode)`
- `setOperation(operation)`
- `setRecords(dupeRecords)`

setEntityType(entityType)

Parameters

- **entityType** {constant} [required] - Set to a constant value defined on the `nlobjDuplicateJobRequest` object. When you pass in the constant, your code should look like `<nlobjDuplicateJobRequestInstance>.<constant>`. The following are the constant values:

- ENTITY_CUSTOMER
- ENTITY_CONTACT
- ENTITY_LEAD
- ENTITY_PROSPECT
- ENTITY_PARTNER
- ENTITY_VENDOR



Note: Note that if you set **entityType** to ENTITY_CUSTOMER, the system will automatically include prospects and leads in the job request.

Returns

- void

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMasterId(masterID)

Parameters

- **masterID** {string} [required] - Required and valid **only** if **setMasterSelectionMode(mode)** is set to MASTERSELECTIONMODE_SELECT_BY_ID

Returns

- void

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMasterSelectionMode(mode)

Parameters

- **mode** {string} [required] - Set to a constant value defined on the **nlobjDuplicateJobRequest** object. When you pass in the constant, your code should look like **<nlobjDuplicateJobRequestInstance>.<constant>**. The following are the constant values:
 - MASTERSELECTIONMODE_CREATED_EARLIEST
 - MASTERSELECTIONMODE_MOST_RECENT_ACTIVITY

- MASTERSELECTIONMODE_MOST_POPULATED_FIELDS
- MASTERSELECTIONMODE_SELECT_BY_ID

Returns

- void

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setOperation(operation)

Parameters

- **operation** {string} [required] - Set to a constant value defined on the **nlobjDuplicateJobRequest** object. When you pass in the constant, your code should look like **<nlobjDuplicateJobRequestInstance>.<constant>**. The following are the constant values:
 - OPERATION_MERGE
 - OPERATION_DELETE
 - OPERATION_MAKE_MASTER_PARENT
 - OPERATION_MARK_AS_NOT_DUPES

Returns

- void

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setRecords(dupeRecords)

Parameters

- **dupeRecords** {Array} [required] - Array of records to be merged

Returns

- void

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjEmailMerger

Encapsulates a scriptable email template, which can be merged with one of the following record types:

- Contact
- Case
- Customer
- Employee
- Partner
- Vendor
- All transaction types
- All custom records

To create a new **nlobjEmailMerger** object, call [nlapiCreateEmailMerger\(templateId\)](#).

See [nlapiCreateEmailMerger\(templateId\)](#) for a sample script.

The nlobjEmailMerger object is supported in all server-side scripts.

Methods

- [merge\(\)](#)
- [setCustomRecord\(recordType, recordId\)](#)
- [setEntity\(entityType, entityId\)](#)
- [setRecipient\(recipientType, recipientId\)](#)
- [setSupportCase\(caseId\)](#)
- [setTransaction\(transactionId\)](#)

merge()

Use this method to perform a mail merge on an **nlobjEmailMerger** object (a scriptable e-mail template) and the records designated with the **nlobjEmailMerger** set methods.

This method has a governance of 20 usage units.

Returns

- An [nlobjMergeResult](#) object containing the e-mail subject and body.

Throws

- `SSS_MERGER_ERROR_OCCURRED` – Thrown if the template merger fails.

Since

- Version 2015 Release 1

setCustomRecord(recordType, recordId)

Use this method to designate a custom record to use in a mail merge.

Parameters

- **recordType** {string} [required] – the internal ID of the custom record type. For example, "customrecord_telco_customer".
- **recordId** {number} [required] – The internal ID of the custom record to use in the mail merge.

Returns

- Void

Throws

- `SSS_INVALID_TYPE_ARG` – Thrown if the recordType argument is invalid or missing.

Since

- Version 2015 Release 1

setEntity(entityType, entityId)

Use this method to designate an entity to use in a mail merge.

Parameters

- **entityType** {string} [required] – The record type of the record to use in the mail merge. Use one of the following arguments:
 - customer
 - contact
 - partner
 - vendor
 - employee
- **entityId** {number} [required] – The internal ID of the record to use in the mail merge

Returns

- Void

Throws

- `SSS_INVALID_TYPE_ARG` – Thrown if the `entityType` argument is invalid or missing.
- `SSS_MERGER_ERROR_OCCURRED` – Thrown if the entity cannot be set.

Since

- Version 2015 Release 1

setRecipient(recipientType, recipientId)

Use this method to designate a second entity (as a recipient) to use in a mail merge.

Parameters

- **recipientType** {string} [required] – The record type of the record to use in the mail merge. Use one of the following arguments:
 - `customer`
 - `contact`
 - `partner`
 - `vendor`
 - `employee`
- **recipientId** {number} [required] – The internal ID of the record to use in the mail merge.

Returns

- `Void`

Throws

- `SSS_INVALID_TYPE_ARG` – Thrown if the `recipientType` argument is invalid or missing.
- `SSS_MERGER_ERROR_OCCURRED` – Thrown if the recipient cannot be set.

Since

Version 2015 Release 1

setSupportCase(caseId)

Use this method to designate a support case to use in a mail merge.

Parameters

- **caseId** {number} [required] – The internal ID of the case record to use in the mail merge.

Returns

- `Void`

Since

Version 2015 Release 1

setTransaction(transactionId)

Use this method to designate a transaction to use in a mail merge. All transaction types are supported

Parameters

- **transactionId** {number} [required] – the internal ID of the transaction record to use in the mail merge.

Returns

- Void

Throws

- SSS_MERGER_ERROR_OCCURRED – Thrown if the transaction cannot be set.

Since

- Version 2015 Release 1

nlobjError

Primary object used to encapsulate errors in the system. Note that the [nlapiCreateError\(code, details, suppressNotification\)](#) function returns a reference to this object.

nlobjError Methods

- [getCode\(\)](#)
- [getDetails\(\)](#)
- [getId\(\)](#)
- [getInternalId\(\)](#)
- [getStackTrace\(\)](#)
- [getUserEvent\(\)](#)

getCode()

Returns the error code for this system or user-defined error

Returns

- The error code as a string

Since

- Version 2008.2

Example

The following script tries to send out an email following the submit of a new record. In the event that an error is thrown, an execution log entry is created and the script continues (user is redirected to the record in EDIT mode).

```
function afterSubmit(type)
{
    if ( type == 'create' )
    {
        try
        {
            var subject = 'A '+nlaiGetRecordType()+ ' with id '+nlapiGetRecordId()+ ' was just created';
            nlapiSendEmail( '-5', 'alerts@company.com', subject );
        }
        catch ( e )
        {
            if ( e instanceof nlobjError )
            {
                nlapiLogExecution( 'DEBUG', 'system error', e.getCode() + '\n' + e.getDetails() )
            }
            else
            {
                nlapiLogExecution( 'DEBUG', 'unexpected error', e.toString() )
            }
            nlapiSetRedirectURL( 'RECORD', nlapiGetRecordType(), nlapiGetRecordId(), true);
        }
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getDetails()

Returns the error message (user-defined or system) associated with this error

Returns

- The string value of the error message

Since

- Version 2008.2

Example

See the sample for [getCode\(\)](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getId()

Returns an error reference ID. If you have included a catch block in your code, you can use **getId()** to get the internal reference number for an unexpected error. This method is useful if you want to keep your own log of error numbers or you want to email the value of **getId()** to someone else.

Also note that if you have to call Customer Support to help you resolve a SuiteScript issue, this ID may be helpful to your Support rep in diagnosing the problem.

Note: If you do not use `getId()` to programmatically get the ID, you can also view the ID in the UI. After a script has executed, the script's error ID (if there is an error) appears on the Execution Log subtab of the Script Deployment page. The ID also appears on the Execution Log subtab in the SuiteScript Debugger. Finally, if you have chosen to be emailed whenever there is a problem with a script, the error ID is provided in the email that is sent to you.

Returns

- The error ID as a string

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getInternalId()

Returns the internal ID of the submitted record if this error was thrown in an **afterSubmit** script

Returns

- The internal ID of the submitted record as an integer

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getStackTrace()

Returns the stacktrace containing the location of the error

Returns

- String[]

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getUserEvent()

Return the name of the user event script (if applicable) that the error was thrown from.

Returns

- The string value of the user event that threw the error - for example, `beforeLoad`, `beforeSubmit`, or `afterSubmit`

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjFile

Primary object used to encapsulate files (media items) in the NetSuite file cabinet. For an example that shows how to use several of the File object methods to upload a file to the NetSuite file cabinet and also attach the file to a record, see [Uploading Files to the File Cabinet Using SuiteScript](#) in the NetSuite Help Center.

nlobjFile Methods

- [getDescription\(\)](#)
- [getFolder\(\)](#)
- [getId\(\)](#)
- [getName\(\)](#)
- [getSize\(\)](#)
- [getType\(\)](#)
- [getURL\(\)](#)
- [getValue\(\)](#)
- [isInactive\(\)](#)
- [isOnline\(\)](#)
- [setDescription\(description\)](#)
- [setEncoding\(encodingType\)](#)
- [setFolder\(id\)](#)
- [setIsInactive\(inactive\)](#)
- [setIsOnline\(online\)](#)
- [setName\(name\)](#)



Note: The following functions return a reference to **nlobjFile**:

- [nlapiCreateFile\(name, type, contents\)](#)
- [nlapiLoadFile\(id\)](#)
- [nlapiMergeRecord\(id, baseType, baseId, altType, altId, fields\)](#)
- [nlapiPrintRecord\(type, id, mode, properties\)](#)

getDescription()

Returns

- The string description of the file. This is the description that appears in the Description field on the folder or file record.

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getEncoding()

Returns the character encoding of a file. NetSuite supports the following encoding types:

- Unicode (UTF-8)
- Western (Windows 1252)
- Western (ISO-8859-1)
- Chinese Simplified (GB 18030)
- Japanese (Shift-JIS)
- Western (Mac Roman)
- Chinese Simplified (GB 2312)
- Chinese Traditional (Big5)

Returns

- One of the following values:
 - UTF-8
 - windows-1252
 - ISO-8859-1
 - GB18030
 - SHIFT_JIS
 - MacRoman
 - GB2312
 - Big5

Since

- Version 2010.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFolder()

Returns

- Integer: The internal ID of the file's folder within the NetSuite file cabinet, for example **10, 2**, etc.

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getId()

Returns the internal ID of the file (if the file is stored in the NetSuite file cabinet)

Returns

- The integer value of file ID, for example **8, 108, 11**, etc. This is the ID that appears in the **Internal ID** column next to the file in the file cabinet.

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getName()

Returns the name of the file

Returns

- The string value of the file name

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSize()

Returns the size of the file in bytes

Returns

- The integer value of the file size

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getType()

Returns the type of the file

Returns

- The string value of the file type - for example, PDF, CSV, PLAINTEXT. (For a list of supported file type IDs, see the help topic [SuiteScript 1.0 Supported File Types](#) .)

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getURL()

Returns the URL to the file if it is stored in the NetSuite file cabinet

Returns

- The URL as a string

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue()

Returns the contents of the file (base 64 encoded for binary files).



Important: This method is only supported on files up to 10MB in size.

Returns

- The string value of the file contents

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isInactive()

Returns

- Boolean: The file's inactive status as either **true** or **false**. Returns true if the file is inactive.

Since

- Version 2009.1

See also

- [setIsInactive\(inactive\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isOnline()

Returns

- Boolean: The file's online status as either **true** or **false**. Returns true if the file is "Available without Login."

Since

- Version 2009.1

See also

- [setIsOnline\(online\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDescription(description)

Sets the description of the file

Parameters

- **description** {string} [required] - A description of the file. This description will appear in the Description field on the folder or file record.

Returns

- void

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setEncoding(encodingType)

Sets the character encoding of a file. The following types are supported when setting the encoding for new and existing files:

- Unicode (UTF-8)
- Western (Windows 1252)
- Western (ISO-8859-1)
- Chinese Simplified (GB 18030)
- Japanese (Shift-JIS)
- Western (Mac Roman)

The following types are supported when setting the encoding for existing files:

- Chinese Simplified (GB 2312)
- Chinese Traditional (Big5)

Parameters

- **encodingType** {string} [required] - The type of encoding for the file. Use one of the following case sensitive values:
 - UTF-8
 - windows-1252
 - ISO-8859-1
 - GB18030
 - SHIFT_JIS
 - MacRoman
 - GB2312
 - Big5



Important: GB2312 and Big5 are not valid arguments when setting the encoding for a new file.

Returns

- void

Since

- Version 2010.1

Example

```
var newFile = nlapiCreateFile('Chinese.csv', 'CSV', csvText);
newFile.setFolder(csvFolderId);
newFile.setEncoding('UTF-8');
nlapiSubmitFile(newFile);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFolder(id)

Sets the internal ID of the folder that the file is in

Parameters

- **id** {int} [required] - The internal ID of the file's folder, for example **10**, **-4**, **20**, etc.

Returns

- void

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setIsInactive(inactive)

Sets the file's inactive status. When you inactive a file or folder, it no longer appears on lists unless (in the UI) you have selected the **Show Inactives** check box.



Note: The Show Inactives check box appears in the bottom-left corner of the Folders list. Navigate to the Folders list by going to Documents > Files > File Cabinet.

Parameters

- inactive** {boolean} [required] - The file's inactive status. Set to *true* to inactive the file. Set to **false** to make the file active.

Returns

- void

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setIsOnline(online)

Sets the file's online ("Available without Login") status. When a file is online, other users can download the file without a login session. This means you can upload images, MP3, or any other file type to the file cabinet and give other users the file URL without giving them access to the account.

Parameters

- online** {boolean} [required] - The file's updated online status. Set to **true** to make the file available online. Set to **false** if you do not want the file available online.

Returns

- void

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setName(name)

Sets the name of the file

Parameters

- **name** {string} [required]- The name of the file

Returns

- void


Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

Uploading Files to the File Cabinet Using SuiteScript

This sample shows how to upload a file into the NetSuite file cabinet. It also shows how to attach this same file to a particular record. See the screenshots after this sample for more details.

 **Note:** The `nlapiSubmitFile` function can submit `nlobjFile` objects of any size, as long as the file size is permitted by the file cabinet.

Example

```
function uploader(request, response)
{
    if (request.getMethod() == 'GET')
    {
        var form = nlapiCreateForm('Attach File to Customer');
        var entityField = form.addField('entity', 'select', 'Customer', 'customer');
        entityField.setLayoutType('normal', 'startcol')
        entityField.setMandatory(true)

        var fileField = form.addField('file', 'file', 'Select File');
        fileField.setMandatory(true)

        form.addSubmitButton();
        form.addResetButton();
        response.writePage(form);
    }
    else
    {
        var entity = request.getParameter("entity")
        var file = request.getFile("file")

        // set the folder where this file will be added. In this case, 10 is the internal ID
        // of the SuiteScripts folder in the NetSuite file cabinet
        file.setFolder(10)

        // Create file and upload it to the file cabinet.
        var id = nlapiSubmitFile(file)

        // Attach file to customer record
```

```

nlobjAttachRecord("file", id, "customer", entity)

// Navigate to customer record
response.sendRedirect('record', 'customer', entity)
}
}

```

The following figure shows the output of this script. To attach a file to a particular customer, specify the customer in the **Customer** field. Next, select a file from the **Select File** field. Click **Save** when finished.




















After clicking Save, you are redirected to the customer record that was specified in the Customer field. In this case, the customer is **Abe Simpson** (see the following figure).

When the Abe Simpson customer record opens, click the Files subtab to verify that the file you selected was attached to the record. In this case, the file is a txt file called **sample file**.

ATTACHED FILES	FOLDER	SIZE (KB)	LAST MODIFIED
sample file	SuiteScripts	1	8.1.2015 12:31 pm

You can also go to the NetSuite file cabinet to verify that **sample file.txt** was uploaded to the SuiteScripts folder. Navigate to the SuiteScripts folder by going to Documents > Files > SuiteScripts.

The following figure shows the **sample text.txt** file in the SuiteScript folder.

SuiteScripts			
EDIT	INTERNAL ID	NAME ▲	SIZE
Edit	3505	 dummygateway.js	15 KB
Edit	3728	 email_capture.js	3 KB
Edit	278	 externalIdText.js	1 KB
Edit	70	 field_changed.js	1 KB
Edit	312	 joinSearch.js	3 KB
Edit	315	 joinSearch_barcode.js	3 KB
Edit	316	 joinSearch_barcode_kr.js	3 KB
Edit	160	 LineItems.js	1 KB
Edit	162	 LineItemsDoNotCommit.js	1 KB
Edit	167	 portletHighOpenBalance.js	2 KB
Edit	179	 preferenceTest.js	1 KB
Edit	271	 processSalesOrders.js	1 KB
Edit	158	 RedirecttoTask.js	2 KB
Edit	168	 resolveURL.js	1 KB
Edit	2983	 RoyalMailPlugin_final.js	10 KB
Edit	322	 sample file.txt	1 KB
Edit	321	 scratch.js	3 KB
Edit	171	 searchCheckBox.js	1 KB
Edit	280	 SetEmailReply.js	1 KB

nlobjFuture

Encapsulates the properties of a merge duplicate record job status. Note that [nlobjManager.getFuture\(\)](#) returns a reference to this object.

nlobjFuture Methods

- [isDone\(\)](#)
- [isCancelled\(\)](#)

isDone()

Returns

- boolean - true if job has finished

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isCancelled()

Returns

- boolean - for merge duplicate records, will always returns false

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjJobManager

Encapsulates the properties of a job manager. A call to [nlapiGetJobManager\(jobType\)](#) returns a reference to this object. Use the methods in **nlobjJobManager** to create and submit your merge duplicate records job request.



Important: When submitting a “merge duplicates” job, the maximum size of your job can be 200 record.

For an end-to-end example that shows how the job manager APIs work together, see [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

nlobjJobManager Methods

- [createJobRequest\(\)](#)
- [submit\(nlobjDuplicateJobRequest\)](#)
- [getFuture\(\)](#)

createJobRequest()

Returns

- [nlobjDuplicateJobRequest](#)

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

submit(nlobjDuplicateJobRequest)

Use to submit your job request. When submitting a “merge duplicates” job, the maximum size of your job can be 200 record.

Be aware that submitting a job places the job into the NetSuite work queue for processing. Submitting a job does not mean that the job is executed right away.

Parameters

- **nlobjDuplicateJobRequest** {Object} [required] - The job you want to submit

Returns

- The jobID is returned if the job is successfully submitted

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFuture()

Use to return a [nlobjFuture](#) object. Then use the methods on the **nlobFuture** object to check the status of the job. Note that a call to **getFuture** costs 5 governance units.

Returns

- [nlobjFuture](#)

Since

- Version 2013.1

Example

See [Example - Using the Job Manager APIs to Merge Duplicate Records](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjLogin

Primary object used to encapsulate NetSuite user login credentials. Note that [nlapiGetLogin\(\)](#) returns a reference to this object.

nlobjLogin Methods

- [changeEmail\(currentPassword, newEmail, justThisAccount\)](#)

- [changePassword\(currentPassword, newPassword\)](#)

changeEmail(currentPassword, newEmail, justThisAccount)

Sets the logged-in user's email address to a new one.

Parameters

- **currentPassword** {string} [required] - The current password of the logged-in user. If a valid value is not specified, an error will be thrown.
- **newEmail** {string} [required] - The new email address for the logged-in user. If a valid value is not specified, an error will be thrown.
- **justThisAccount** {boolean} [optional] - If not set, this argument defaults to **true**. If set to **true**, the email address change is applied only to roles within the current account. If set to **false**, the email address change is applied to all accounts and roles.

Since

- Version 2012.2

Example

This example shows how to change the logged-in user's email address.

```
//Get the logged-in user's credentials
var login = nlapiGetLogin();
//Change current email address
login.changeEmail('MyCUrl3ntPa$$word', 'newemail@netsuite.com', true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

changePassword(currentPassword, newPassword)

Sets the logged-in user's password to a new one.

Parameters

- **currentPassword** {string} [required] - The current password of the logged-in user. If a valid value is not specified, an error will be thrown.
- **newPassword** {string} [required] - The new password for the logged-in user. If a valid value is not specified, an error will be thrown.

Since

- Version 2012.2

Example

This example shows how to change the logged-in user's password.

```
//Get the currently logged-in user credentials
```

```
var login = nlapiGetLogin();  
//Change current password  
login.changePassword('MyUrr3ntPa$word', 'MyNewPaSw0rD!');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjMergeResult

The nlobjMergeResult object is supported in all server-side scripts.

Methods

- [getBody\(\)](#)
- [getSubject\(\)](#)

getBody()

Use this method to get the body of the email distribution in string format.

Returns

- A string

Since

Version 2015 Release 1

getSubject()

Use this method to get the subject of the email distribution in string format.

Returns

- A string

Since

Version 2015 Release 1

nlobjPivotColumn

Object used to encapsulate a pivot table column.

Methods

- [getAlias\(\)](#)

- [getParent\(\)](#)
- [getLabel\(\)](#)
- [getSummaryLine\(\)](#)
- [getValue\(\)](#)
- [getVisibleChildren\(\)](#)
- [isHidden\(\)](#)

getAlias()

Get the column alias.

Returns

- string - The column alias.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getDependency(alias)

Returns

getParent()

Get the parent column.

Returns

- [nlobjPivotColumn](#) - Null if it does not exist

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLabel()

Get the column label.

Returns

- string - Column label

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSummaryLine()

Get the summary line.

Returns

- [nlobjPivotColumn](#) - Summary line if it exists, otherwise null

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue()

Get the value of the column.

Returns

- object - The value of this column

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getVisibleChildren()

Get any defined children columns.

Returns

- [nlobjPivotColumn](#)[] - Null if no children columns exist

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isHidden()

Checks if the column is hidden.

Returns

- boolean - True if the column is hidden

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjPivotRow

Object used to encapsulate a pivot table row.

Methods

- [getAlias\(\)](#)
- [getChildren\(\)](#)
- [getLabel\(\)](#)
- [getOpeningLine\(\)](#)
- [getParent\(\)](#)
- [getSummaryLine\(\)](#)
- [getValue\(\)](#)
- [getValue\(pivotColumn\)](#)
- [isDetailLine\(\)](#)

getAlias()

Get the row alias.

Returns

- string - The row alias.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getChildren()

Get the children rows if there are any.

Returns

- [nlobjPivotRow](#)[] - Null if the row is a detail line or if there are no children.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

Get the row label.

- string - The row label.

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getOpeningLine()

Returns

Since

getParent()

Get the summary line from the report.

Returns

- [nlobjPivotRow](#) - Null if the row does not exist.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSummaryLine()

Get the parent row if it exists.

Returns

- [nlobjPivotRow](#) - Null if the row is a detail line.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue()

Get the row value if the row is a detail line.

Returns

- object - The value of the row hierarchy, or null if **isDetailLine** returns false.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue(pivotColumn)

Get the value of the row/column combination.

Parameters

- **pivotColumn** {[nlobjPivotColumn](#)} [required] - The pivot column.

Returns

- object - The value of the row/column combination, or null if **isDetailLine** returns false.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isDetailLine()

Check if the row is a detail line.

Returns

- boolean - True if the row is a detail line.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjPivotTable

Object used to encapsulate the pivot table.

Methods

- [getColumnHierarchy\(\)](#)
- [getRowHierarchy\(\)](#)

getColumnHierarchy()

Get the column hierarchy.

Returns

- [nlobjPivotColumn](#)

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRowHierarchy()

Get the row hierarchy.

Returns

- [nlobjPivotRow](#)

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjPivotTableHandle

Handle to the pivot table object. A handle is a reference which points to the pivot table.

Methods

- [getPivotTable\(\)](#)
- [isReady\(\)](#)

getPivotTable()

Get the pivot table object from the report definition.



Note: This is a blocking call and it will wait until the report definition execution has finished. Using **isReady** is recommended to check execution state if blocking is unacceptable.

Returns

- [nlobjPivotTable](#)

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isReady()

Returns the completion status flag of the report definition execution.

Returns

- boolean - True if the execution has finished.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjRecord


Primary object used to encapsulate a NetSuite 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](#). For help working with an instance of a custom record type, see the help topic [Custom Record](#).

Methods

- [calculateTax\(\)](#)
- [commitLineItem\(group, ignoreRecalc\)](#)
- [createCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [createSubrecord\(fldname\)](#)
- [editCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [editSubrecord\(fldname\)](#)
- [findLineItemMatrixValue\(group, fldnam, column, val\)](#)
- [findLineItemValue\(group, fldnam, value\)](#)
- [getAllFields\(\)](#)
- [getAllLineItemFields\(group\)](#)
- [getCurrentLineItemDateTimeValue\(type, fieldId, timeZone\)](#)
- [getCurrentLineItemMatrixValue\(group, fldnam, column\)](#)
- [getCurrentLineItemValue\(type, fldnam\)](#)
- [getCurrentLineItemValues\(type, fldnam\)](#)
- [getDateTimeValue\(fieldId, timeZone\)](#)
- [getField\(fldnam\)](#)
- [getFieldText\(name\)](#)
- [getFieldTexts\(name\)](#)
- [getFieldValue\(name\)](#)
- [getFieldValues\(name\)](#)

- getId()
- getLineItemCount(group)
- getLineItemDateTimeValue(type, fieldId, lineNum, timeZone)
- getLineItemField(group, fldnam, linenum)
- getLineItemMatrixField(group, fldnam, linenum, column)
- getLineItemMatrixValue(group, fldnam, linenum, column)
- getLineItemText(group, fldnam, linenum)
- getLineItemValue(group, name, linenum)
- getLineItemValues(type, fldnam, linenum)
- getMatrixCount(group, fldnam)
- getMatrixField(group, fldname, column)
- getMatrixValue(group, fldnam, column)
- getRecordType()
- insertLineItem(group, linenum, ignoreRecalc)
- removeLineItem(group, linenum, ignoreRecalc)
- removeCurrentLineItemSubrecord(sublist, fldname)
- removeSubrecord(fldname)
- selectLineItem(group, linenum)
- selectNewLineItem(group)
- setCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)
- setCurrentLineItemMatrixValue(group, fldnam, column, value)
- setCurrentLineItemValue(group, name, value)
- setDateTimeValue(fieldId, dateTime, timeZone)
- setFieldText(name, text)
- setFieldTexts(name, text)
- setFieldValue(name, value)
- setFieldValues(name, value)
- setLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)
- setLineItemValue(group, name, linenum, value)
- setMatrixValue(group, fldnam, column, value)
- viewCurrentLineItemSubrecord(sublist, fldname)
- viewLineItemSubrecord(sublist, fldname, linenum)
- viewSubrecord(fldname)

 **Note:** The following functions return a reference to the `nlobjRecord` object:

- nlapiCopyRecord(type, id, initializeValues)
- nlapiCreateRecord(type, initializeValues)
- nlapiGetNewRecord()
- nlapiGetOldRecord()
- nlapiLoadRecord(type, id, initializeValues)

- `nlobjTransformRecord(type, id, transformType, transformValues)`

calculateTax()

Supported in server script in dynamic mode when the SuiteTax feature is enabled in the account. Calculates tax in a particular record. Tax calculation results are stored within the record. Line-level tax amounts are also available in tax details. This list is not scriptable but is available in the UI and through searches. Taxes are always calculated when the transaction is saved.

`nlobjRecord` is the primary object used to encapsulate a NetSuite record.

Parameters

- None

Returns

- void

Since

- 2015.2

Throws

- `SSS_UNSUPPORTED_METHOD`, if the SuiteTax feature is disabled, or the transaction type of the current record is non-taxable
- `SSS_TAX_REGISTRATION_REQUIRED`, if the subsidiary of the current record does not have a valid tax registration

Example

```
var salesOrder = nlobjCreateRecord('salesorder', {recordmode: 'dynamic'});
salesOrder.setFieldValue('entity', 1234);
salesOrder.selectNewLineItem('item')
salesOrder.setCurrentLineItemValue('item', 'item', 1234);
salesOrder.commitLineItem('item');
salesOrder.calculateTax();

// do something with the calculated values in salesOrder
```

commitLineItem(group, ignoreRecalc)

Use this method to commit the current line in a sublist.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use *addressbook* as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

- **ignoreRecalc** {Boolean true|false} [optional] – If set to true, the total is not recalculated upon execution. Use this parameter if you are editing multiple line items on the same sublist and you need to improve performance. Do not use this option on the last commit of the sublist; the last `commitLineItem` call must recalculate the total. An error is thrown upon record submit if you do not recalculate the total on the last `commitLineItem` of the sublist. This parameter is only supported with server-side scripts.

Returns

- void

Since

- Version 2009.2

Example

This sample shows how to create a new Vendor Bill record and then add items to the Item sublist and expenses to the Expenses sublist. Note that because you are adding new lines to each sublist, you must call the `selectNewLineItem(group)` method. You then set all values for the new lines using the `setCurrentLineItemValue(group, name, value)` method. When you are finished adding values to each sublist, you must commit all sublist updates using the `commitLineItem(group)` method.

```
var record = nlapiCreateRecord('vendorbill');
record.setFieldValue('entity', 196);
record.setFieldValue('department', 3);
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'item', 380);
record.setCurrentLineItemValue('item', 'location', 102);
record.setCurrentLineItemValue('item', 'amount', '2');
record.setCurrentLineItemValue('item', 'customer', 294);
record.setCurrentLineItemValue('item', 'isbillable', 'T');
record.commitLineItem('item');

record.selectNewLineItem('expense');
record.setCurrentLineItemValue('expense', 'category', 3);
record.setCurrentLineItemValue('expense', 'account', 11);
record.setCurrentLineItemValue('expense', 'amount', '10');
record.setCurrentLineItemValue('expense', 'customer', 294);
record.setCurrentLineItemValue('expense', 'isbillable', 'T');
record.commitLineItem('expense');

var id = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

createCurrentLineItemSubrecord(sublist, fldname)

Returns a `nlobjSubrecord` object. Use this API to create a subrecord from a **sublist field** on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Creating an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

createSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to create a subrecord from a **body field** on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Creating an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

editCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to edit a subrecord from a **sublist** field on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Editing an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

editSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to edit a subrecord from a **body** field on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2


Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

findLineItemMatrixValue(group, fldnam, column, val)

Use this method to return the line number of a particular price in a specific column. If the value is present on multiple lines, it will return the line item of the **first** line that contains the value.

Use this API on a matrix sublists only.

 **Note:** Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.
- **val** {string} [required] - The value of the field

Returns

- The line number (as an integer) of a specified matrix field

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

findLineItemValue(group, fldnam, value)

Use this API to return the line number for the first occurrence of a field value in a sublist column. This API can be used on any sublist type that supports SuiteScript (editor, inline editor, and list sublists).

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The field internal ID
- **value** {string} [required] - The value of the field

Returns

- The line number (as an integer) of a specific sublist field

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllFields()

Returns a normal keyed array of all the fields on a record. Note that the number of fields returned will differ when you call `getAllFields()` on the edit of a record vs. on the xedit of a record. For details, see these topics :

- [Inline Editing and nlapiGetNewRecord\(\)](#)
- [Inline Editing and nlapiGetOldRecord\(\)](#)
- [What's the Difference Between xedit and edit User Event Types?](#)

Returns

- String[] of all field names on the record

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllLineItemFields(group)

Returns an array of all the field names of a sublist on this record

Parameters

- **group** {string} [required]- The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- String[] of sublist field names

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCurrentLineItemDateTimeValue(type, fieldId, timeZone)

Returns the value of a datetime field on the currently selected line of a sublist. If timeZone is passed in, the datetime value is converted to that time zone and then returned. If timeZone is not passed in, the datetime value is returned in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. This field ID must point to a datetime formatted field.
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table. If this argument is not supplied, the time zone will default to the time zone set in user preferences.

Returns

- The string value of a datetime field on the currently selected line.

Throws

- `SSS_INVALID_ARG_TYPE`

Since

- Version 2013 Release 2

Example

```
var a = nlapiLoadRecord('customrecord_parentdatetime', 1);
a.selectLineItem('recmachcustrecord_childdatetime', 1);
var tz = a.getCurrentLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimeetcol', 'America/Regina');
```

getCurrentLineItemMatrixValue(group, fldnam, column)

Use this API to get the value of the currently selected matrix field. This API should be used on matrix sublists only.



Important: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field being set.
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.

Returns

- The string value of a field on the currently selected line in a matrix sublist. Returns **null** if the field does not exist.

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCurrentLineItemValue(type, fldnam)

Returns the value of a sublist field on the currently selected line

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use `addressbook` as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field being set

Returns

- The string value of a field on the currently selected line. Returns null if field does not exist.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCurrentLineItemValues(type, fldnam)

Returns the values of a multiselect sublist field on the currently selected line. One example of a multiselect sublist field is the Serial Numbers field on the Items sublist.

This function is not supported in client SuiteScript. It is meant to be used in user event scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use `addressbook` as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the multiselect field

Returns

- An array of string values for the multiselect sublist field

Since

- Version 2012.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getDateTimeValue(fieldId, timeZone)

Returns the value of a datetime field. If `timeZone` is passed in, the datetime value is converted to that time zone and then returned. If `timeZone` is not passed in, the datetime value is returned in the default time zone.

Parameters

- **fieldId** {string} [required] — The internal field ID. This field ID must point to a datetime formatted field.
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- The string value of a datetime field.

Throws

- `SSS_INVALID_ARG_TYPE`

Since

- Version 2013 Release 2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getField(fldnam)

Returns field metadata for a field. This method is only supported with server-side scripts.

Parameters

- **fldnam** {string} [required] - The internal ID of the field

Returns

- The [nlobjField](#) object

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldText(name)

Returns the UI display value for a select field. This method is only supported with server-side scripts. This method is supported on select fields only.

Parameters

- **name** {string} [required] - The internal ID of the field

Returns

- String UI display value corresponding to the current selection for a select field. Returns *null* if field does not exist on the record or if the field is restricted.

Since

- Version 2009.1

Example

The sample below shows how to use **getFieldText(name)**. In this sample, the script will return the UI display value of the Sales Rep (salesrep) field. In this account, the Sales Rep has been set to **Abe Simpson**. This is the value that will be returned.

```
var rec = nlapiLoadRecord('salesorder', 1957);
var valText = rec.getFieldText('salesrep'); // returns Abe Simpson
```


See also

- [nlapiGetFieldText\(fldnam\)](#) - this is the form-level client-side equivalent of `nlobjRecord.getFieldText(name)`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldTexts(name)

Returns the UI display values for a multi-select field. This method is only supported with server-side scripts. This method is supported on multi-select fields only.

Parameters

- **name** {string} [required] - The internal ID of the multiselect field

Returns

- String[] - Returns the selected text values of a multi-select field

Since

- Version 2009.1

Example

The sample below shows how to use `getFieldTexts(name)`. In this sample, the script will return the UI display values of a custom multiselect field that references customers in the account. The internal ID for the multiselect field is `custbody23`. In this account, the multiselect field has the display values of **104 Lou Liang** and **105 Barry Springsteen**. These are the values that will be returned.

```
var rec = nlapiLoadRecord('salesorder', 1957); // load the sales order
var valText = rec.getFieldTexts('custbody23'); // returns 104 Lou Liang and 105 Barry Springsteen
```

See also

[nlapiGetFieldTexts\(fldnam\)](#) - this is the form-level client-side equivalent of `nlobjRecord.getFieldTexts(name)`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValue(name)

Returns the value (internal ID) of a field.

Note that NetSuite recommends you read the topic [Getting Field Values in SuiteScript](#), which addresses the rare instances in which the value returned by this API is inconsistent.

Parameters

- **name** {string} [required] - The internal ID of the field whose value is being returned.

Returns

- The internal ID (string) value for the field

Example

In this sample, the script returns the internal ID of the value in the Partner (partner) field. In this particular sales order, the Partner field has been set to ABC Inc., which has an internal ID value of 219. The value 219 will be returned in this script.

```
var rec = nlapiLoadRecord('salesorder', 18); // load a sales order
var value = rec.getFieldValue('partner'); // get internal ID value of the Partner field
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValues(name)


Returns the value (field ID) or values (array of field IDs) of a multi-select field.

Parameters

- **name** {string} [required]- The name of the field whose value is being returned

Returns

- If there is only one value selected in the multi-select field, this method returns the field ID as a string.
- If there are multiple values selected in the multi-select field, this method returns a string array of field IDs.
- If the field is not on the record, this method returns **null**.

 **Note:** To determine whether **getFieldValues** returns a string or an array, compare the return value to the return value of **nlobjRecord.getFieldValue**. The **getFieldValue** method returns a string.

Example

In this sample, the script returns an array of internal ID values that are set in a custom multi-select field called Advertising Preferences. (In this account, the internal ID of the Advertising Preferences field is custentity1.)

In the UI, the Advertising Preferences field has the values of E-mail and Mail. The internal ID values for E-mail and Mail are 2 and 3, respectively. The values of 2 and 3 will be returned in this script.

```
var rec = nlapiLoadRecord('customer', 196); // load a customer record
var values = rec.getFieldValues('custentity1'); //get array of internal ID values set in custentity1 field
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getId()

Use this method to get the internal ID of a record or NULL for new records.

Returns

- Integer value of the record ID

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemCount(group)

Returns the number of lines on a sublist



Important: The first line number on a sublist is **1** (not 0).

Parameters

- **group** {string} [required]- The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- The integer value of the number of line items on a sublist

getLineItemDateTimeValue(type, fieldId, lineNum, timeZone)

Returns the value of a datetime field on a sublist. If `timeZone` is passed in, the datetime value is converted to that time zone and then returned. If `timeZone` is not passed in, the datetime value is returned in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **lineNum** {int} [required] — The line number for this field. Note the first line number on a sublist is 1 (not 0).
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- The string value of a datetime field on a sublist.

Throws

- `SSS_INVALID_ARG_TYPE`

Since

- Version 2013 Release 2

Example

```
var a = nlapiLoadRecord('customrecord_parentdatetime', 1);
```

```
var tz = a.getLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzcol', 1, 'America/Regina');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemField(group, fldnam, linenum)

Returns field metadata for a line item (sublist) field. This method is only supported with server-side scripts.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use *addressbook* as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the line item field
- **linenum** {int} [required] - The line number this field is on. Note the first line number on a sublist is **1** (not 0). Only settable for sublists of type **list**.

Returns

- An [nlobjField](#) object

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemMatrixField(group, fldnam, linenum, column)

Use this API to obtain metadata for a field that appears in a matrix sublist.



Important: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the field (line) whose value you want returned.
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- **column** {int} [required] - The column number for this field. Column numbers start at 1, not 0.

Returns

- An [nlobjField](#) object representing this sublist field. Returns *null* if the field you have specified does not exist.

Since

- Version 2009.2

Example

```
record = nlapiLoadRecord('inventoryitem', 312);
var itemid = record.getFieldValue('itemid');
//Get the metadata for the price matrix field.
var matrixFieldObj = record.getLineItemMatrixField('price1', 'price', 1, 2);
var fieldLabel = matrixFieldObj.getLabel();
var fieldName = matrixFieldObj.getName();
var fieldType = matrixFieldObj.getType();

var fieldMetaInfo = 'Label: '+fieldLabel+' Name: '+fieldName+' Type: '+fieldType ;
record.setFieldValue('purchasedescription', fieldMetaInfo);

var id2 = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemMatrixValue(group, fldnam, lineum, column)

Use this API to get the value of a matrix field that appears on a specific line in a specific column. This API can be used only in the context of a matrix sublist.

Note: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Parameters

- group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- fldnam** {string} [required] - The internal ID of the matrix field whose value you want returned.
- lineum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- The string value of the matrix field

Since

- Version 2009.2

Example

```
record = nlapiLoadRecord('inventoryitem', 333);
var itemid = record.getFieldValue('itemid');
```

```
var price1 = record.getLineItemMatrixValue('price1', 'price', 1, 1);
var price2 = record.getLineItemMatrixValue('price1', 'price', 2, 1);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemText(group, fldnam, linenum)

Returns the display name of a select field (based on its current selection) in a sublist. This method is only supported with server-side scripts.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The name of the field/line item being set
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- String - The string UI display value corresponding to the current selection for a line item select field. Returns **null** if field does not exist on the record or the field is restricted.

Since

- Version 2009.1

Example

The sample below shows how to set `getLieItemText(type, fldnam, linenum)`. In this sample, the script will return the UI display name value of the Item (item) field on the Item sublist. In this account, the Item field has been set to **Assorted Bandages**. This is the value that will be returned.

```
var rec = nlapiLoadRecord('salesorder', 1957);
var valText = rec.getFieldText('salesrep');
var line1txt= rec.getLineItemText('item', 'item', 1);
```

See also

- [nlapiGetLineItemText\(type, fldnam, linenum\)](#) - this is the form-level client-side equivalent of `nlobjRecord.getLineItemText`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemValue(group, name, linenum)

Returns the value of a sublist line item field.

Note that NetSuite recommends you read the topic [Getting Field Values in SuiteScript](#), which addresses the rare instances in which the value returned by this API is inconsistent.

Note: Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **name** {string} [required]- The name of the sublist field whose value is being returned
- **linenum** {int} [required]- The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of the sublist field name

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemValues(type, fldnam, linenum)

Returns the values of a multiselect sublist field on a selected line. One example of a multiselect sublist field is the Serial Numbers field on the Items sublist.

This function is not supported in client SuiteScript. It is meant to be used in user event scripts.

Parameters

- **type** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the multiselect field
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- An array of string values for the multiselect sublist field


Since


- Version 2012.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getMatrixCount(group, fldnam)

Use this API in a matrix sublist to get the number of columns for a specific matrix field.

 **Important:** Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

 **Note:** The first column in a matrix is 1, not 0.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The field internal ID of the matrix field.

Returns

- The integer value for the number of columns of a specified matrix field

Since

- Version 2009.2


Example

```
record = nlapiLoadRecord('inventoryitem', 333);
var itemid = record.getFieldValue('itemid');
var count = record.getMatrixCount('price', 'price');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getMatrixField(group, fldname, column)

Use this API to get field metadata for a matrix “header” field in a matrix sublist. This method is only supported with server-side scripts.

 **Important:** Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

For example, if the Quantity Pricing feature is enabled in your account, you will see the **Qty** fields at the top of the pricing matrix. The Qty fields are considered to be the header fields in the pricing matrix. For more information on matrix header fields, see the help topic [Matrix APIs](#) in the NetSuite Help Center.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix header field.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- [nlobjField](#) object

Since

- Version 2009.2

Example

This sample shows how to get the metadata of the quantity (Qty) field on the USA Pricing tab.

```
record = nlapiLoadRecord('inventoryitem', 333);
var itemid = record.getFieldValue('itemid');

//Get the metadata of quantity field inside the USA Pricing tab
var fieldObj = record.getMatrixField('prices1', 'price', 1);
var fieldLabel = fieldObj.getLabel();
var fieldName = fieldObj.getName();
var fieldType = fieldObj.getType();

var fieldMetaInfo = 'Label: '+fieldLabel+' Name: '+fieldName+' Type: '+fieldType ;
record.setFieldValue('purchasedescription', fieldMetaInfo);
var id2 = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getMatrixValue(group, fldnam, column)

Use this API to get the value of a matrix “header” field in a matrix sublist.



Important: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

For example, if the Quantity Pricing feature is enabled in your account, you will see the **Qty** fields at the top of the pricing matrix. The Qty fields are considered to be the header fields in the pricing matrix. See the help topic [Matrix APIs](#) in the NetSuite Help Center for more information on matrix header fields.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix header field.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).

Returns

- The string value of a matrix header field

Since

- Version 2009.2

Example

```
record = nlapiLoadRecord('inventoryitem', 333);
var itemid = record.getFieldValue('itemid');
var quant1 = record.getMatrixValue('price1', 'price', '2');
record.setFieldValue('purchasedescription', quant1);
var id2 = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRecordType()

Returns the record type (for example assembly *unbuild* would be returned for the Assembly Unbuild record type; *salesorder* would be returned for the Sales Order record type).

Returns

- The string value of the record name internal ID

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

insertLineItem(group, linenum, ignoreRecalc)

Inserts a new line into a sublist. This function is only supported for edit sublists (inlineeditor, editor). Note, however, this API will work on *list* sublists that have been added via the UI object nlobjSubList

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **linenum** {int} [required] - Line index at which to insert the line. Note that in sublists, the first line number is **1** (not 0). If the number is greater than the number of lines on the sublist, an error is returned.
- **ignoreRecalc** {Boolean true|false} [optional] – If set to true, the total is not recalculated upon execution. Use this parameter if you are inserting multiple line items on the same sublist and you need to improve performance. Do not use this option on the last line item insert of the sublist; the last **insertLineItem** call must recalculate the total. An error is thrown upon record submit if you do not recalculate the total on the last insertLineItem of the sublist. This parameter is only supported with server-side scripts.

Returns

- void

Example

```
// insert line at the beginning of the item sublist
var rec = nlapiGetNewRecord();
rec.insertLineItem('item', 1);
```

```

rec.setLineItemValue('item', 'quantity', 1, 10);

// insert line at the end
// triggered in the beforeSubmit event
var rec = nlapiGetNewRecord();
var intCount = rec.getLineItemCount('item');

rec.insertLineItem('item', intCount + 1);
rec.setLineItemValue('item', 'quantity', intCount + 1, 10);

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

removeLineItem(group, linenum, ignoreRecalc)

Use this method to remove an existing line from a sublist.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- **ignoreRecalc** {Boolean true|false} [optional] - If set to true, the total is not recalculated upon execution. Use this parameter if you are removing multiple line items on the same sublist and you need to improve performance. Do not use this option on the last line item removal of the sublist; the last **removeLineItem** call must recalculate the total. An error is thrown upon record submit if you do not recalculate the total on the last **removeLineItem** of the sublist. This parameter is only supported with server-side scripts.

Returns

- void

Since

- Version 2009.2

Example

```

for (j=1; j <= soRecord.getLineItemCount('item'); j++)
{
    soRecord.removeLineItem('item', '1');
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

removeCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to remove a subrecord from a **sublist** field on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- void

Since

- Version 2011.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

removeSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to remove a subrecord from a **body** field on the parent record.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- void

Since

- Version 2011.2

Example

See the help topic [Removing an Inventory Detail Subrecord from a Sublist Line](#) in the NetSuite Help Center.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

selectLineItem(group, linenum)

Use this method to select an existing line in a sublist.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- void

Since

- Version 2009.2

Example

```
var record = nlapiCreateRecord('inventoryitem');
record.setFieldValue('itemid', '124');
record.setFieldValue('department', 3);
record.setMatrixValue('price1', 'price', '2', 500);

record.selectLineItem('price', '1');
record.setCurrentLineItemMatrixValue('price', 'price', 1, '100');
record.setCurrentLineItemMatrixValue('price', 'price', 2, '90');
record.commitLineItem('price');

var id = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

selectNewLineItem(group)

Use this method to insert and select a new line in a sublist.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- void

Since

- Version 2009.2

Example

This sample shows how to create a new Vendor Bill record and then add items to the Item sublist and expenses to the Expenses sublist. Note that because you are adding **new** lines to each sublist, you

must call the `selectNewLineItem(group)` method. You then set all values for the new lines using the `setCurrentLineItemValue(group, name, value)` method. When you are finished adding values to each sublist, you must commit all sublist updates using the `commitLineItem(group, ignoreRecalc)` method.

```
var record = nlapiCreateRecord('vendorbill');
record.setFieldValue('entity', 196);
record.setFieldValue('department', 3);
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'item', 380);
record.setCurrentLineItemValue('item', 'location', 102);
record.setCurrentLineItemValue('item', 'amount', '2');
record.setCurrentLineItemValue('item', 'customer', 294);
record.setCurrentLineItemValue('item', 'isbillable', 'T');
record.commitLineItem('item');

record.selectNewLineItem('expense');
record.setCurrentLineItemValue('expense', 'category', 3);
record.setCurrentLineItemValue('expense', 'account', 11);
record.setCurrentLineItemValue('expense', 'amount', '10');
record.setCurrentLineItemValue('expense', 'customer', 294);
record.setCurrentLineItemValue('expense', 'isbillable', 'T');
record.commitLineItem('expense');

var id = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)

Sets the value of a datetime field on the currently selected line of a sublist. If `timeZone` is passed in, the datetime value is converted to that time zone and then set. If `timeZone` is not passed in, the datetime value is set in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am | pm (for example, '09/25/2013 06:00:01 am').
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

Example

```
var a = nlapiLoadRecord('customrecord_parentdatetime', 1);
a.selectLineItem('recmachcustrecord_childdatetime', 1);
a.setCurrentLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzcol', '01/01/2013
06:00:01 am', 'America/Phoenix');
a.commitLineItem('recmachcustrecord_childdatetime');
nlapiSubmitRecord(a);
```

setCurrentLineItemMatrixValue(group, fldnam, column, value)

Use this API to set the value of a matrix sublist field. Also note that it should be used on matrix sublists only.



Important: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The internal ID of the matrix field.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).
- **value** {string | int} [required] - The value the field is being set to.

Returns

- void

Since

- Version 2009.2

Example

```
var record = nlapiCreateRecord('inventoryitem');
record.setFieldValue('itemid', '124');
record.setFieldValue('department', 3);
record.setMatrixValue('price1', 'price', '2', 500);

record.selectLineItem('price', '1');
record.setCurrentLineItemMatrixValue('price', 'price', 1, '100');
record.setCurrentLineItemMatrixValue('price', 'price', 2, '90');
record.commitLineItem('price');
```

```
var id = nlapiSubmitRecord(record, true);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setCurrentLineItemValue(group, name, value)

Use this method to set the value of a sublist line item field.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **name** {string} [required] - The name of the field being set
- **value** {string} [required] - The value the field is being set to.



Important: Check box fields take the values of **T** or **F**, not **true** or **false**.

Returns

- void

Since

- Version 2009.2

Example

This sample shows how to create a new Vendor Bill record and then add items to the Item sublist and expenses to the Expenses sublist. Note that because you are adding new lines to each sublist, you must call the [selectNewLineItem\(group\)](#) method. You then set all values for the new lines using the [setCurrentLineItemValue\(group, name, value\)](#) method. When you are finished adding values to each sublist, you must commit all sublist updates using the [commitLineItem\(group, ignoreRecalc\)](#) method.

```
var record = nlapiCreateRecord('vendorbill');
record.setFieldValue('entity', 196);
record.setFieldValue('department', 3);
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'item', 380);
record.setCurrentLineItemValue('item', 'location', 102);
record.setCurrentLineItemValue('item', 'amount', '2');
record.setCurrentLineItemValue('item', 'customer', 294);
record.setCurrentLineItemValue('item', 'isbillable', 'T');
record.commitLineItem('item');

record.selectNewLineItem('expense');
record.setCurrentLineItemValue('expense', 'category', 3);
record.setCurrentLineItemValue('expense', 'account', 11);
record.setCurrentLineItemValue('expense', 'amount', '10');
record.setCurrentLineItemValue('expense', 'customer', 294);
record.setCurrentLineItemValue('expense', 'isbillable', 'T');
record.commitLineItem('expense');
```



```
var id = nlapiSubmitRecord(record, true);
```

setDateTimeValue(fieldId, dateTime, timeZone)

Sets the value of a datetime field. If timeZone is passed in, the datetime value is converted to that time zone and then set. If timeZone is not passed in, the datetime value is set in the default time zone.

Parameters

- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am|pm (for example, '09/25/2013 06:00:01 am').
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldText(name, text)

Sets the value of a select field using its corresponding display value. This method is only supported with server-side scripts.

Parameters

- **name** {string} [required] - The internal ID of the field being set
- **text** {string} [required] - The display value corresponding to the value the field is being set to

Returns

- void

Since

- Version 2009.1

Example

```
var record = nlapiLoadRecord('salesorder', 1955); // load the sales order
```

```
record.setFieldText('location', 'East Coast'); // set the field display value for Location to East Coast
var id = nlapiSubmitRecord(record, true); // submit the record
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldTexts(name, text)

Sets the values for a multiselect field from their display values. This method is only supported with server-side scripts.

Parameters

- **name** {string} [required] - The internal ID of the field being set
- **texts** {string[]} [required] - The display values corresponding to the values the field is being set to

Returns

- void

Since

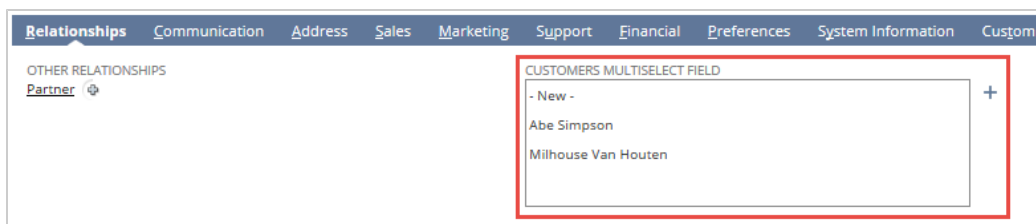
- Version 2009.1

Example

```
var values = new Array(); // create an array of customers who are currently in NetSuite
values[0] = 'Abe Lincoln'; // add the first customer
values[1] = 'Abe Simpson'; // add the second customer
var record = nlapiLoadRecord('salesorder', 447); // load the sales order

// set the field display values for the custom multiselect field
// called Customers Multiselect Field
record.setFieldTexts('custbody16', values);

// submit the record
var submit = nlapiSubmitRecord(record, true);
```



[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValue(name, value)

Sets the value of a field

Parameters

- **name** {string} [required] - The name of the field being set
- **value** {string} [required] - The value the field is being set to

Returns

- void

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValues(name, value)

Sets the value of a multi-select field

Parameters

- **name** {string} [required] - The name of the field being set
- **value** {string[]} [required]- String array containing field values

Returns

- void

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)

Sets the value of a datetime field on a sublist. If timeZone is passed in, the datetime value is converted to that time zone and then set. If timeZone is not passed in, the datetime value is set in the default time zone.

Parameters

- **type** {string} [required] — The internal sublist ID
- **fieldId** {string} [required] — The internal field ID. The field ID passed in must point to a datetime formatted field.
- **lineNum** {int} [required] — The line number for this field. Note the first line number on a sublist is 1 (not 0).
- **dateTime** {string} [required] — The date and time in format mm/dd/yyyy hh:mm:ss am|pm (for example, '09/25/2013 06:00:01 am').
- **timeZone** {string | int} [optional] — If a string is passed in, it must match one of the Olson Values listed in the [SuiteScript 1.0 Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [SuiteScript 1.0 Olson Values](#) table.

Returns

- void

Throws

- SSS_INVALID_ARG_TYPE

Since

- Version 2013 Release 2

Example

```
var a = nlapiLoadRecord('customrecord_parentdatetime', 1);
a.setLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzcol', 1, '01/01/2013 06:00:01 am', 'America/Phoenix');
nlapiSubmitRecord(a);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLineItemValue(group, name, linenum, value)

Sets the value of a sublist line item.

Note: Normally custom transaction column fields that are not checked to show on a custom form are not available to **get/setLineItemValue** APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **name** {string} [required] - The name of the field being set
- **linenum** {int} [required] - The line number for this field. Note the first line in a sublist is **1** (not 0).
- **value** {string} [required] - The value the field is being set to. If a valid value is not specified an error will be thrown.

Returns

- void

Since

- Version 2008.1

Example

The following example shows how to create a new record and then add a sublist to the record. In this case a Partner sublist is being added to a newly created Sale Order.

```
/*Create a Sales Order record. Next, add a field to the record and then add an *item, which must be added before a Sales Order can be saved.
```

```

*/
var record = nlapiCreateRecord('salesorder');
record.setFieldValue('entity', 87);
record.setLineItemValue('item','item', 1, 458);
record.setFieldValue('shippingcost',12);

/*Add a Partners sublist to the Sales Order. Note you must provide a valid value
*for the Partner ID. In this case, to obtain Partner IDs you can look in the UI
*under Lists > Relationships > Partners. Ensure that the Show Internal ID
*preference is enabled. IDs will appear in the ID column of the Partner list.
*/
record.setLineItemValue('partners','partner', 1,311);
record.setLineItemValue('partners','partnerrole', 1,1);
record.setLineItemValue('partners', 'isprimary',1, 'T' );
record.setLineItemValue('partners', 'contribution',1, 100 );

//Finally, submit the record to save it.
var id = nlapiSubmitRecord(record, true);

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMatrixValue(group, fldnam, column, value)

This API is used to set a header field in a matrix sublist. Also note that this API should be used on matrix sublists only.



Important: Currently the Pricing sublist is the only matrix sublist type that supports SuiteScript. For details on working with the Pricing sublist, see the help topic [Pricing Sublist / Pricing Matrix](#) in the NetSuite Help Center.

In the case of the Pricing sublist, this API is used to set the quantity levels that appear in the Qty fields. Note that you should use this API only if you have the Quantity Pricing feature enabled in your account, as these header fields appear only if this feature is enabled.

Parameters

- **type** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see the help topic [Pricing Sublist Internal IDs](#) to determine the correct internal ID of your pricing list.
- **fldnam** {string} [required] - The name of the field being set.
- **column** {int} [required] - The column number for this field. Column numbers start at 1 (not 0).
- **value** {string} [required] - The value the field is being set to.



Important: Check box fields take the values of T or F, not **true** or **false**.

Returns

- void

Since

- Version 2009.2

Example

The following sample shows how to set pricing matrix values on a new Inventory Item record. In this sample, **setMatrixValue** is used to set the quantity levels in Qty columns 2, 3, 4, 5. Note that in this account, the Multi-Currency feature has been enabled and all pricing matrix values are being set on the USA pricing tab (price1).

```
var record = nlapiCreateRecord('inventoryitem');
record.setFieldValue('itemid', '124');
record.setFieldValue('department', 3);
record.setMatrixValue('price1', 'price', '2', 500);
record.setMatrixValue('price1', 'price', '3', 600);
record.setMatrixValue('price1', 'price', '4', 700);
record.setMatrixValue('price1', 'price', '5', 800);
//Now set prices to all pricelevel and quantity level fields on the USA tab.
//Set Base prices in different columns.
record.selectLineItem('price1', '1');
record.setCurrentLineItemMatrixValue('price1', 'price', 1, '100');
record.setCurrentLineItemMatrixValue('price1', 'price', 2, '200');
record.setCurrentLineItemMatrixValue('price1', 'price', 3, '300');
record.setCurrentLineItemMatrixValue('price1', 'price', 4, '400');
record.setCurrentLineItemMatrixValue('price1', 'price', 5, '500');

record.commitLineItem('price1');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

viewCurrentLineItemSubrecord(sublist, fldname)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **sublist** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in **read-only** mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this API.

You can call this API when you want your script to read the **nlobjSubrecord** object of the current sublist line you are on.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use **item** as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).

Returns

- **nlobjSubrecord**

Since

- Version 2011.2

Example

See the help topic [Viewing an Inventory Detail Subrecord](#) in the NetSuite Help Center.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

viewLineItemSubrecord(sublist, fldname, linenum)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **sublist** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in read-only mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this function.

You can call this API when you want to read the value of a line you are **not** currently on. For example, if you are editing line 2, you can call this API on line 1 to get the value of line 1.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **sublist** {string} [required] - The sublist internal ID on the parent record (for example, use *item* as the ID for the Items sublist).
- **fldname** {string} [required] - The internal ID of the “subrecord field” on the sublist of the parent record (for example, **inventorydetail** as the ID for the Inventory Details sublist field).
- **linenum** {int} [required] - The line number for the sublist field. Note the first line number on a sublist is 1 (not 0).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

viewSubrecord(fldname)

Returns a **nlobjSubrecord** object. Use this API to view a subrecord from a **body** field on the parent record. Calling this API analogous to doing a “get” on a subrecord, however, the **nlobjSubrecord** object returned is in read-only mode. Therefore, an error is thrown if you attempt to edit a subrecord returned by this function.

See the help topic [Working with Subrecords in SuiteScript](#) for general information on working with subrecords in NetSuite.

Parameters

- **fldname** {string} [required] - The internal ID of the “subrecord field” on the body of the parent record (for example, **inventorydetail** as the ID for the Inventory Details body field).

Returns

- [nlobjSubrecord](#)

Since

- Version 2011.2

Example

See the help topic [Viewing an Inventory Detail Subrecord](#) in the NetSuite Help Center.

nlobjReportColumn

Object used to encapsulate a report column on a pivot report.

Methods

- [getFormula\(\)](#)
- [getParent\(\)](#)
- [isMeasure\(\)](#)

getFormula()

Get the formula for this column

Returns

- string - Formula or null if it does not exist.

getParent()

Get the parent reference of this column.

Returns

- The parent reference to the [nlobjReportColumnHierarchy](#) object.

isMeasure()

Returns the measure flag

Returns

- boolean - True if the column is flagged as a measure.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjReportColumnHierarchy

Object used to encapsulate the report column hierarchy.

Methods

- [getChildren\(\)](#)
- [getParent\(\)](#)

getChildren()

Get the children reference of this column hierarchy.

Returns

- The child reference to the [nlobjReportColumnHierarchy](#) object.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getParent()

Get the parent reference of this column hierarchy.

Returns

- Either the parent reference to the [nlobjReportColumnHierarchy](#) object or null.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjReportDefinition

The primary object that contains the definition of the report. For an example that shows how to use several of the **nlobjReportDefinition** object methods to programmatically render a pivot table report in a browser, see [Building a Pivot Report Using SuiteScript](#).

Methods

- [addColumn\(alias, isMeasure, label, parent, format, formula\)](#)
- [addColumnHierarchy\(alias, label, parent, format\)](#)
- [addRowHierarchy\(alias, label, format\)](#)
- [addSearchDatasource\(searchType, id, filters, columns, map\)](#)
- [executeReport\(form\)](#)

- [setTitle\(title\)](#)

addColumn(alias, isMeasure, label, parent, format, formula)

Add a column to the report definition.

Parameters

- **alias** {string} [required] - The column alias.
- **isMeasure** {boolean} [required] - A value of true means that the column is flagged as a measure.
- **label** {string} [required] - The column label that will be displayed on the report.
- **parent** {[nlobjReportColumnHierarchy](#)} [optional] - The reference to the parent column in the hierarchy. If null, then this column will not be associated with a parent column.
- **format** {string} [required] - The data type that this column represents.
- **formula** {string} [optional] - A string which describes a mathematical formula in the format of "F(x,y,z) = mathematical function", where x,y,z are previously defined aliases from [addRowHierarchy](#), [addColumnHierarchy](#), or [addColumn](#) calls.

Returns

- The reference to the [nlobjReportColumn](#) object.

Since

- Version 2012.2

Example

See the code sample in [Building a Pivot Report Using SuiteScript](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addColumnHierarchy(alias, label, parent, format)

Add a column hierarchy to the report definition.

Parameters

- **alias** {string} [required] - The column alias.
- **label** {string} [required] - The column label that will be displayed on the report.
- **parent** {[nlobjReportColumnHierarchy](#)} [optional] - The reference of the parent column in the hierarchy. If null, then this column will be the root (top level) column.
- **format** {string} [required] - The data type that this column represents.

Returns

- The reference to the [nlobjReportColumnHierarchy](#) object.

Since

- Version 2012.2

Example

See the code sample in [Building a Pivot Report Using SuiteScript](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRowHierarchy(alias, label, format)

Add a row hierarchy to the report definition.

Parameters

- **alias** {string} [required] - The row alias.
- **label** {string} [required] - The row label that will be displayed on the report.
- **format** {string} [required] - The data type that this row represents.

Returns

- The reference to the [nlobjReportRowHierarchy](#) object.

Since

- Version 2012.2

Example

See the code sample in [Building a Pivot Report Using SuiteScript](#).


[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSearchDatasource(searchType, id, filters, columns, map)

Attaches a search as a data source to the report definition.

Parameters

- **searchType** {string} [required] - The type of records to search.
- **id** {string} [optional] - The internal id (as a string) if you are using a saved search as a data source.
- **filters** {[nlobjSearchFilter\[\]](#)} [required] - The array of search filters.

 **Note:** Search filter expression as **filters** parameter is currently not supported.

- **columns** {[nlobjSearchColumn\(name, join, summary\)\[\]](#)} [required] - The array of search columns.
- **map** {string} [required] - The mapping of rows/columns of the search to the report.

Since

- Version 2012.2

Example

This snippet of code shows how a data source is set up. Observe how the columns are mapped.

```
var reportDefinition = nlapiCreateReportDefinition();

var columns = new Array();
var filters = new Array();

columns[0] = new nlobjSearchColumn('internalID', null, 'group');
columns[1] = new nlobjSearchColumn('entity', null, 'group');
filters[0] = new nlobjSearchFilter('status', null, 'anyof', 'InProgress');

reportDefinition.addSearchDataSource('opportunity', null, filters, columns,
  {'internalID':columns[0], 'entity':columns[1]});
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

executeReport(form)

Creates the form for rendering from the report definition.

Parameters

- form** {[nlobjReportForm](#)} [optional] - The form object created with [nlapiCreateReportForm](#).
 If not specified the call waits until the execution is finished (synchronous) and an [nlobjPivotTable](#) will be available from the handle. If the parameter is set, the call returns immediately and the returned value references the [nlobjReportForm](#) - a pivot table handle will not be available in this case.

Note: Only one synchronous pivot table execution is allowed at a time. If a second synchronous execution is called, it will invalidate the first pivot table.

Returns

- [nlobjPivotTableHandle](#) - The identifier of the pivot table handle, or [nlobjReportForm](#).

Since

- Version 2012.2

Example 1

This example shows how to create a pivot table for basic rendering as a report in a browser.

```
//Create a form to put the report on
var myForm = nlapiCreateReportForm('Pivot Report Sales Orders');

//Populate form here
...

//Build the form from the report definition
var myReportForm = reportDefinition.executeReport(myForm);
```

```
//Write the form back to the browser
response.writePage(myReportForm);
```

Example 2

This example shows how to create a pivot table for further processing with SuiteScript. The pivot table is not rendered.

```
//Create a form to put the report on
var myform = nlapiCreateReportForm('Pivot Report Sales Orders');

//Populate form here
...

//Build the form from the report definition, get the pivot table handle
var myPivotTableHandle = reportDefinition.executeReport();

//Get the pivot table object
var myPivotTable = myPivotTableHandle.getPivotTable();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setTitle(title)

Sets the title of the report definition.

Parameters

- **title** {string} [optional] - The name of the report definition.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

Building a Pivot Report Using SuiteScript

This example shows how to create a report showing the opportunities for each customer, and opportunity status. Each opportunity status is broken down to show the projected total and the probability of each opportunity.

Use the method **executeReport** passing along an optional **form** parameter rather than void so that the form definition is built onto a standard [nlobjReportForm](#) object that can be rendered on the browser using the **response.writePage** method.

```
function runOpportunitiesPivot(request, response)
{
    //Instantiate a report definition to work with
    var reportDefinition = nlapiCreateReportDefinition();

    //Define the rows/column hierarchy and the actual column data
    var customer = reportDefinition.addRowHierarchy('entity', 'Customer', 'TEXT');
```

```

var salesrep= reportDefinition.addColumn('salesrep', false, 'Sales Rep', null, 'TEXT', null);
var entstat = reportDefinition.addColumnHierarchy('entitystatus', 'Opportunity Status', null, 'TEXT');
var total = reportDefinition.addColumn('projectedtotal', true, 'Projected Total',
    entstat, 'CURRENCY', null);
var prob = reportDefinition.addColumn('probability', false, 'Probability %', entstat, 'PERCENT', null);

//Create the search to feed the report
var columns = new Array();
columns[0] = new nlobjSearchColumn('internalID', null, 'group');
columns[1] = new nlobjSearchColumn('entity', null, 'group');
columns[2] = new nlobjSearchColumn('salesrep', null, 'group');
columns[3] = new nlobjSearchColumn('expectedclosedate', null, 'group');
columns[4] = new nlobjSearchColumn('entitystatus', null, 'group');
columns[5] = new nlobjSearchColumn('projectedtotal', null, 'sum');
columns[6] = new nlobjSearchColumn('probability', null, 'group');

//Add search to the report and map the search columns to the reports columns
var filters = new Array();
filters[0] = new nlobjSearchFilter('projectedtotal', null, 'greaterthan', 2000);
reportDefinition.addSearchDataSource('opportunity', null, filters, columns,
    {'internalID':columns[0], 'entity':columns[1], 'salesrep':columns[2], 'expectedclosedate':columns[3],
    'entitystatus':columns[4], 'projectedtotal':columns[5], 'probability':columns[6]});

//Create a form to build the report on
var form = nlapiCreateReportForm('Pivot Report Suitelet: Opportunities');

//Build the form from the report definition
var pvtTable = reportDefinition.executeReport(form);

//Write the form to the browser
response.writePage(form);
}

```

The following figure shows how the pivot report example is rendered in the NetSuite UI.

Note: Right-click and open in New Tab to see full-sized image.

Pivot Report Suitelet: Opportunities														
CUSTOMER	SALES REP	QUALIFIED		PROPOSAL		IDENTIFIED DECISION MAKERS		CLOSED WON		PURCHASING		IN NEGOTIATION		TOTAL
		Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total
■ Aaron Rosewall-Godley														
	Mathew Christner	\$3,692.00												\$3,692.00
	Mathew Christner			\$26,644.20										\$26,644.20
Total - Aaron Rosewall-Godley		\$3,692.00		\$26,644.20		\$0.00		\$0.00		\$0.00		\$0.00		\$30,336.20
Abdullah Bhupathiraju	Luke Duke	\$3,692.00												\$3,692.00
Adam Fitzpatrick	Jon Baker					\$5,000.00								\$5,000.00
Adam Kaveimacher	J Wolfe	\$2,300.00												\$2,300.00
Adelina Shonkiewicz	Mathew Christner	\$14,849.66												\$14,849.66
Adina Fitzpatrick	J Wolfe			\$4,300.00										\$4,300.00
Andrew Goldwasser	Jon Baker							\$14,478.15						\$14,478.15
Apu Nahasapeemepetion	Jon Baker	\$18,861.80												\$18,861.80
Barney Brooking	Jon Baker							\$8,544.00						\$8,544.00
Barney Gumble	Jessie Barto							\$3,692.00						\$3,692.00
Beverly Linden	Theodore Hosch							\$8,532.75						\$8,532.75
■ Billings Dental Clinic														
	Christian Begum									\$7,384.00				\$7,384.00
	Theodore Hosch									\$4,293.90				\$4,293.90
	Theodore Hosch	\$6,330.75												\$6,330.75
	Theodore Hosch	\$4,660.88												\$4,660.88
	J Wolfe											\$5,100.00		\$5,100.00
Total - Billings Dental Clinic		\$10,991.63		\$0.00		\$0.00		\$0.00		\$11,677.90		\$5,100.00		\$27,769.53

In SuiteScript, this looks like:

```

var customer = reportDefinition.addRowHierarchy('entity', 'Customer', 'TEXT');

```

```
var salesrep= reportDefinition.addColumn('salesrep', false, 'Sales Rep', null, 'TEXT', null);
var entstat = reportDefinition.addColumnHierarchy('entitystatus', 'Opportunity Status', null, 'TEXT');
var total = reportDefinition.addColumn('projectedtotal', true, 'Projected Total', entstat, 'CURRENCY', null);
var prob = reportDefinition.addColumn('probability', false, 'Probability %', entstat, 'PERCENT', null);
```

nlobjReportForm

Object used to encapsulate the report form and render the report in HTML.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjReportRowHierarchy

Object used to encapsulate the report row hierarchy.

Methods

- [getChild\(\)](#)
- [getParent\(\)](#)

getChild()

Get the child reference of this row hierarchy.

Returns

- The child reference to the [nlobjReportRowHierarchy](#) object.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getParent()

Get the parent reference of this row hierarchy.

Returns

- Either the parent reference to the [nlobjReportRowHierarchy](#) object or null.

Since

- Version 2012.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjRequest

Primary object used to encapsulate an HTTP GET or POST request. When creating Suitelets, you pass **request** and **response** arguments to your user-defined function (see example). With the request object instantiated, you can then call any **nlobjRequest** method.

Example

```
function demoSimpleForm(request, response)
{
    //call an nlobjRequest method
    if (request.getMethod() == 'GET')
    {
        var form = nlapiCreateForm('Simple Form');
        //remainder of code...

        response.writePage(form);
    }
}
```

nlobjRequest Methods

- [getAllHeaders\(\)](#)
- [getAllParameters\(\)](#)
- [getBody\(\)](#)
- [getFile\(id\)](#)
- [getHeader\(name\)](#)
- [getLineItemCount\(group\)](#)
- [getLineItemValue\(group, name, line\)](#)
- [getMethod\(\)](#)
- [getParameter\(name\)](#)
- [getParameterValues\(name\)](#)
- [getSSPURL\(\)](#)
- [getURL\(\)](#)

getAllHeaders()

Returns an Array containing all of the request headers and their values.

 **Note:** For security reasons, JSESSIONID headers are not returned.

Returns

- Associative Array of header name/value pairs. This array typically includes two iterations of each name/value pair: one with the name represented in lower case, and one with the name in title case.

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllParameters()

Returns an Object containing all the request parameters and their values

Returns

- String[] of parameter field names

Since

- Version 2008.2

Example

The following example shows how to set or read multiple parameters from a request object by iterating through the properties of the object

```
var params = request.getAllParameters()
for ( param in params )
{
    nlapiLogExecution('DEBUG', 'parameter: '+ param)
    nlapiLogExecution('DEBUG', 'value: '+params[param])
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getBody()

Returns the body of the POST request

Returns

- The string value of the request body

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFile(id)

Returns a file reference (**nlobjFile** object) added to a Suitelet page with the **nlobjForm.addField(name, type, label, sourceOrRadio, tab)** method (where 'file' is passed in as the **type** argument).

10

Returns

- [nlobjFile](#)

Since

- Version 2010.1

Example

See [Uploading Files to the File Cabinet Using SuiteScript](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getHeader(name)

Returns the value of a request header.

Parameters

- **name** {string} [required] - The name of the header. This input is handled in a case-insensitive manner.

Returns

- The header value as a string

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemCount(group)

Returns the number of lines in a sublist



Important: The first line number on a sublist is **1** (not 0).

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See [Using the SuiteScript Records Browser](#) for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.

Returns

- The integer value of the number of line items in a sublist

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemValue(group, name, line)

Returns the value of a sublist line item.

Note: Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **name** {string} [required] - The name of the field whose value is returned
- **line** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of the line item

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getMethod()

Returns the METHOD of the request.

Returns

- The string value of the request type. Request types include GET or POST.

Since

- Version 2008.1

Example

```
function demoSimpleForm( request, response)
{
    if ( request.getMethod() == 'GET' )
    {
        var form = nlapiCreateForm('Simple Form');

        //remainder of code...

        response.writePage(form);
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getParameter(name)

Returns the value of the request parameter

Parameters

- **name** {string} [required]- The name of the request parameter whose value is returned

Returns

- The string value of the request parameter

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getParameterValues(name)

Returns the values of a request parameter as an Array

Parameters

- **name** {string} [required] - The name of the request parameter whose value is returned

Returns

- String[] of parameter values


Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSSPURL()

Returns the full URL of the request for SSP files

 **Note:** This method should be used instead of the [getURL\(\)](#) method.

Returns

- The string value of the request URL for SSP files.

Since

- Version 2015.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getURL()

Returns the full URL of the request



Important: This method might not work correctly for some SSP applications. Use the [getSSPURL\(\)](#) method instead.

Returns

- The string value of the request URL

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjResponse

Primary object used for scripting web responses in Suitelets. Note that the [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#) function returns a reference to this object.

When creating Suitelets you will pass **request** and *response* arguments to your user-defined function (see example). With the response object instantiated, you can then call any **nlobjResponse** method.

See the help topic [SuiteScript 1.0 Supported File Types](#) in the NetSuite Help Center for a list of all content/media types that can be returned through the **nlobjResponse** object.



Note: **nlobjResponse** currently supports only gzip and deflate compression algorithms.

Example

```
function demoSimpleForm(request, response )
{
    if ( request.getMethod() == 'GET' )
    {
        var form = nlapiCreateForm('Simple Form');

        //remainder of code...

        //call the nlobjResponse object writePage method
        response.writePage( form);
    }
}
```

nlobjResponse Methods

- [addHeader\(name, value\)](#)
- [getAllHeaders\(\)](#)
- [getBody\(\)](#)
- [getCode\(\)](#)

- `getError()`
- `getHeader(name)`
- `getHeaders(name)`
- `renderPDF(xmlString)`
- `setCDNCacheable(type)`
- `setContentType(type, name, disposition)`
- `setEncoding(encodingType)`
- `setHeader(name, value)`
- `sendRedirect(type, identifier, id, editmode, parameters)`
- `write(output)`
- `writeLine(output)`
- `writePage(pageobject)`

addHeader(name, value)

Adds a header to the response. If this header has already been set, this will add a new header to the response. Note that all user-defined headers must be prefixed with **Custom-Header** otherwise an `SSS_INVALID_ARG` error will be thrown ()

Parameters

- **name** {string} [required] - The name of the header
- **value** {string} [required] - The value used to set header

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllHeaders()

Returns an Array containing all the headers returned in the response. Only available in the return value of a call to `nlapiRequestURL(url, postdata, headers, callback, httpMethod)`.

Returns

- String[] of headers

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getBody()

Returns the body returned by the server. Only available in the return value of a call to [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#).

 **Note:** `nlobjResponse` currently supports only gzip and deflate compression algorithms.

Returns

- The string value of the body

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCode()

Returns the response code returned by the server. Only available in the return value of a call to [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#).

Returns

- The string value of the response code

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getError()

Returns the [nlobjError](#) thrown during request. Only available in the return value of call to `nlapiRequestURL` in Client SuiteScript.

Returns

- `nlobjError`

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getHeader(name)

Returns the value for a header returned in the response. Only available in the return value of a call to [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#).

Parameters

- **name** {string} [required] - The header name

Returns

- The string value of the header

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getHeaders(name)

Returns an Array containing all the values for a header returned in the response. This is only available in the return value of a call to `nlobjRequestURL`.

Parameters

- **name** {string} - The header name

Returns

- String[] of header values

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

renderPDF(xmlString)

Generates, and renders, a PDF directly to a response. Use **renderPDF** to generate PDFs without first importing a file to the file cabinet. This method is useful if your script does not have NetSuite file cabinet permissions.

The **renderPDF** method uses the Big Faceless Report Generator built by Big Faceless Organization (BFO). The BFO Report Generator is a third-party library used for converting XML to PDF documents. The **renderPDF** method passes XML to the BFO tag library (which is stored by NetSuite), and renders a PDF directly to a response. Note that the **xmlString** argument is the same input string as that passed to BFO by `nlobjXMLToPDF(xmlString)`.

For details on BFO, available tags, and BFO examples, see the following links:

- <http://faceless.org/products/report/docs/userguide.pdf>
- <http://faceless.org/products/report/docs/tags/>



Note: SuiteScript developers do not need to install any BFO-related files or components to use the Report Generator functionality.

The **renderPDF** method is supported in server-side scripts. It has a governance of 10 usage units.

Parameters

- **xmlString** {string} [required] – Content of your PDF, passed to **renderPDF** as a string.

Returns

- void

Since

- Version 2014 Release 2

Example

```
function testSimpleXML(request, response)
{
    var xml = '<?xml version="1.0"?>\n<!DOCTYPE pdf PUBLIC "-//big.faceless.org//report" "report-1.1.dtd">\n<pdf>\n<body font-size="18">\nTesting!\n</body>\n</pdf>';
    response.renderPDF(xml);
}
```

setCDNCacheable(type)

Sets CDN caching for a shorter period of time or a longer period of time. There is no ability to invalidate individual assets, so SSP Application can set its TTL (Time To Live) in CDN and fall into one of four categories:

- Unique — This asset is not cached.
- Short — This asset may change frequently, so cache it for five minutes.
- Medium — This asset may or may not change frequently, so cache it for two hours.
- Long — This asset is not expected to change frequently, so cache it for seven days.



Important: This method is not accessible through a Suitelet. It must be accessed in the context of a shopping SSP.

Parameters

- **type** {constant} [required]- Constant value to represent the caching duration:
 - CACHE_DURATION_UNIQUE
 - CACHE_DURATION_SHORT
 - CACHE_DURATION_MEDIUM
 - CACHE_DURATION_LONG

Note that when setting constant values, you do not use quotation marks. The syntax will be something similar to:

```
setCDNCacheable( response.CACHE_DURATION_SHORT);
```

Returns

- void

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setContenttype(type, name, disposition)

Sets the content type for the custom responses (and an optional file name for binary output). This API is available in Suitelet scripts only.

Parameters

- **type** {string} [required] - The content/file type. For a list of supported file types, see the help topic [SuiteScript 1.0 Supported File Types](#) in the NetSuite Help Center.
- **name** {string} [optional] - Set the name of the file being downloaded (for example 'name.pdf')
- **disposition** {string} [optional] - Content disposition to use for file downloads. Available values are **inline** or **attachment**. If a value is not specified, the parameter will default to **attachment**. What this means is that instead of a new browser (or Acrobat) launching and rendering the content, you will instead be asked if you want to download and Save the file.

Returns

- void

Since

- Version 2008.2

Example

See [Example 2](#) for **nlapixMLToPDF**. This sample shows how to set a file content type to PDF and then, by specifying **inline** as the disposition type, having the PDF open in Acrobat.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setEncoding(encodingType)

Sets the character encoding on **nlobjResponse** content. Available encoding types are:

- Unicode (UTF-8)
- Western (Windows 1252)
- Western (ISO-8859-1)
- Chinese Simplified (GB 18030)
- Chinese Simplified (GB 2312)
- Japanese (Shift-JIS)
- Western (Mac Roman)

The default encoding type is Unicode (UTF-8).

Your browser character encoding settings must match the specified encoding to view the file contents correctly.

Parameters

- **encodingType** {string} [required] - The type of encoding for the response. Use one of the following case sensitive values:
 - UTF-8
 - windows-1252
 - ISO-8859-1
 - GB18030
 - GB2312



Important: GB2312 is not a valid argument when setting the encoding for a new file.

- ▣ SHIFT_JIS
- ▣ MacRoman

Returns

- void

Since

- Version 2013.1

Example

This example shows how to set the encoding of an existing windows-1252 file that has a file ID of 215.

```
var nlFile = nlapiLoadFile('215');
response.setEncoding('windows-1252');
nlapiLogExecution('DEBUG', 'Content', nlFile.getValue());
response.write(nlFile.getValue());
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHeader(name, value)

Sets the value of a response header. Note that all user-defined headers must be prefixed with **Custom-Header** otherwise an SSS_INVALID_ARG or SSS_INVALID_HEADER error will be thrown.



Important: This method is available only in Suitelets.

Parameters

- **name** {string} [required] - The name of the header
- **value** {string} [required] - The value used to set header

Returns

- void

Since

- Version 2008.2

Example

```
function demoHTML(request, response)
{
  var html = '<html><body><h1>Hello World</h1></body></html>';
  response.write( html );
}
```

```
//set a custom header
response.setHeader('Custom-Header-Demo', 'Demo');
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

sendRedirect(type, identifier, id, editmode, parameters)

Sets the redirect URL by resolving to a NetSuite resource. Note that all parameters must be prefixed with **custparam** otherwise an SSS_INVALID_ARG error will be thrown.

Also note that all URLs must be internal unless the Suitelet is being executed in an “Available without Login” context. If this is the case, then within the “Available without Login” (externally available) Suitelet, you can set the **type** parameter to **EXTERNAL** and the **identifier** parameter to the external URL.



Note: If you want to redirect a user to an external URL, you must use this function in a Suitelet and set the **type** parameter to EXTERNAL. See the documentation for the **type** parameter below.



Important: To avoid **414 - Request URI Too large** or **502 — Bad Gateway** errors, limit the length of the URI to less than 2000 characters for all http requests. Different browsers and servers enforce different limits on URI length.

Parameters

- **type** {string} [required] - The base type for this resource
 - **RECORD** - Record Type
 - **TASKLINK** - Task Link
 - **SUITELET** - Suitelet
 - **EXTERNAL** - Custom URL (external) and only available for external Suitelets (i.e. available without login)
- **identifier** {string} [required] - The primary id for this resource (record type ID for RECORD, scriptId for SUITELET, taskId for tasklink, url for EXTERNAL)
- **id** {string} [optional] - The secondary id for this resource (record type ID for RECORD, deploymentId for SUITELET)
- **editmode** {boolean **true** || **false**} [optional] - For RECORD calls, this 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, otherwise the record is returned in view mode.
- **parameters** {hashtable} [optional] - An associative array of additional URL parameters as name/value pairs

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

write(output)

Write information (text/xml/html) to the response

Parameters

- **output** {string | [nlobjFile](#) object} [required] - String or file being written

Returns

- void

Example

```
function demoHTML(request, response)
{
  var html = '<html><body><h1>Hello World</h1></body></html>';
  response.write(html);
  response.setHeader('Custom-Header-Demo', 'Demo');
}
```

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

writeLine(output)

Write line information (text/xml/html) to the response

Parameters

- **output** {string} [required] - String being written

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

writePage(pageobject)

Generates a page using a page element object ([nlobjForm](#) or [nlobjList](#))

Parameters

- **pageobject** {[nlobjForm](#) | [nlobjList](#)} [required] - Standalone page object: [nlobjForm](#) or [nlobjList](#)

Returns

- void

Since

- Version 2008.2

Example

```
function demoSimpleForm(request, response)
{
    if ( request.getMethod() == 'GET' ) {
        var form = nlapiCreateForm('Simple Form');

        //remainder of code...

        response.writePage(form);
    }
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSearch

Primary object used to encapsulate a NetSuite saved search. Note, however, you are **not required** to save the search results returned in this object.

A reference to **nlobjSearch** is returned by [nlapiCreateSearch\(type, filters, columns\)](#) and [nlapiLoadSearch\(type, id\)](#). If you are creating a new search using **nlapiCreateSearch**, the search will not be saved until you call **nlobjSearch.saveSearch(title, scriptId)**.

After you have saved the search, you can get properties of the search or redefine the search by loading the search with [nlapiLoadSearch\(type, id\)](#) and calling various methods on the **nlobjSearch** object. You can also do this for searches created in the UI.

By default, the search returned by **nlapiCreateSearch** will be private, which follows the saved search model in the UI. To make a search public, you must call **nlobjSearch.setIsPublic(type)** before saving it.



Note: You can see the filters and columns properties of **nlobjSearch** in the SuiteScript Debugger after the object is loaded.

For general information on executing NetSuite searches using SuiteScript, see the help topic [SuiteScript 1.0 Searching Overview](#) in the NetSuite Help Center.

Methods

- [addColumn\(column\)](#)
- [addColumns\(columns\)](#)
- [addFilter\(filter\)](#)
- [addFilters\(filters\)](#)
- [deleteSearch\(\)](#)
- [getColumns\(\)](#)

- `getFilterExpression()`
- `getFilters()`
- `getId()`
- `getIsPublic()`
- `getScriptId()`
- `getSearchType()`
- `runSearch()`
- `saveSearch(title, scriptId)`
- `setColumns(columns)`
- `setFilterExpression(filterExpression)`
- `setFilters(filters)`
- `setIsPublic(type)`
- `setRedirectURLToSearch()`
- `setRedirectURLToSearchResults()`

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addColumn(column)

Adds a single return column to the search. Note that existing columns on the search are not changed.

Parameters

- **column** {`nlobjSearchColumn(name, join, summary)`} [required] - The `nlobjSearchColumn` you want added to the search.

Returns

- `void`

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add an additional column. After the new column is added, a new script ID is assigned to the search.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define return columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create the saved search
```

```

var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the existing search and add a new column to the search
var newSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Call addColumn to add column to the existing search
var newColumn = new nlobjSearchColumn('somecolumn');
newSearch.addColumn(newColumn);
var newId = newSearch.saveSearch('My New Search', 'customsearch_kr2');

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addColumnns(columns)

Adds multiple return columns to the search. Note that existing columns on the search are not changed.

Parameters

- **columns** {nlobjSearchColumn(name, join, summary)[]} [required] - The nlobjSearchColumn[] you want added to the search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add columns. After the new columns are added, a new script ID is assigned to the search.

```

// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', '-5', null );
// Define return columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create the saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the existing search
var newSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Define additional columns for the existing search
var newColumns = new Array();
columns[0] = new nlobjSearchColumn('somecolumn');
columns[1] = new nlobjSearchColumn('somecolumn1');
columns[2] = new nlobjSearchColumn('somecolumn2');
// Call addColumns to add columns to the existing search
newSearch.addColumns(newColumns);

```



```
var newId = newSearch.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addFilter(filter)

Adds a single search filter. Note that existing filters on the search are not changed.

Note: This method does not accept a search filter expression (`Object[]`) as parameter. Only a single search filter (`nlobjSearchFilter`) is accepted.

Parameters

- **filter** {`nlobjSearchFilter`} [required] - The `nlobjSearchFilter` you want added to the search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add an additional filter. After the new filter is added, a new script ID is assigned to the search.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define return columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create the saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the existing search and add a new filter to the search
var newSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Call addFilter to add an additional filter to the existing search
var newFilter = new nlobjSearchFilter('somefilter');
newSearch.addFilter(newFilter);
var newId = newSearch.saveSearch('My New Search', 'customsearch_kr2');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addFilters(filters)

Adds a search filter list. Note that existing filters on the search are not changed.

Note: This method does not accept a search filter expression (**Object[]**) as parameter. Only a search filter list (**nlobjSearchFilter[]**) is accepted.

Parameters

- **filters** **nlobjSearchFilter[]** [required] - The list (array) of zero or more **nlobjSearchFilter** you want added to the search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search and then load the search to add filters. After the new filters are added, a new script ID is assigned to the search.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define return columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create the saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the existing search
var newSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Define additional filters to the existing search
var newFilters = new Array();
newFilters[0] = new nlobjSearchFilter('somefilter');
newFilters[1] = new nlobjSearchFilter('somefilter1');
newFilters[2] = new nlobjSearchFilter('somefilter2');
// Call addFilters to add filters to the existing search
newSearch.addFilters(newFilters);
var newId = newSearch.saveSearch('My New Search', 'customsearch_kr2');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

deleteSearch()

Deletes a saved search that was created through scripting or through the UI.

If you have created a saved search through the UI, you can load the search using **nlapiLoadSearch(type, id)** and then call **deleteSearch** to delete it.

In scripting if you have created a search using `nlapiCreateSearch(type, filters, columns)` and saved the search using the `nlobjSearch.saveSearch(title, scriptId)`, you can then load the search and call `deleteSearch` to delete it.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and delete it.

```
// Load the existing search and then delete it
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
mySearch.deleteSearch();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getColumns()

Gets the search return columns for the search.

Returns

- `nlobjSearchColumn(name, join, summary)[]`

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and get its search return columns and its filters.

```
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
var columns = s.getColumns();
var filters = s.getFilters();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFilterExpression()

Gets the filter expression for the search.

Returns

- Object[]

Since

- Version 2012.2

Example

This example shows how to load an existing saved search and get its search filter expression.

```
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
var filterExpression = s.getFilterExpression();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFilters()

Gets the filters for the search.

Note: This method does not return a search filter expression (**Object**[]). Only a search filter list (**nlobjSearchFilter**[]) is returned. If you want to get a search filter expression, see [getFilterExpression\(\)](#).

Returns

- **nlobjSearchFilter**[]

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and get its search return columns and its filters.

```
var s = nlapiLoadSearch('opportunity', 'customsearch_kr');
var columns = s.getColumns();
var filters = s.getFilters();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getId()

Gets the internal ID of the search. The internal ID is available only when the search is either loaded using [nlapiLoadSearch\(type, id\)](#) or has been saved using [nlobjSearch.saveSearch\(title, scriptId\)](#).

If this is an on demand search (created with [nlapiCreateSearch\(type, filters, columns\)](#)), this method will return **null**.

Returns

- The search ID as a string. Typical return values will be something like 55 or 234 or 87. You will not receive a value such as **customsearch_mysearch**. Any ID prefixed with **customsearch** is considered a script ID, not the search's internal system ID.

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and get its internal system ID.

```
// Load the existing search and then get its internal ID assigned by NetSuite
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
var internalId = mySearch.getId();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getIsPublic()

Gets whether the **nlobjSearch** has been set as public search.

Returns

- Returns true if the search is public. Returns false if it is not.

Since

- Version 12.1

Example

This example shows how to load an existing saved search and check whether the search is public or private.

```
// Load the existing search and see if it is public
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
if (mySearch.getIsPublic());
{
    // mySearch is public.
}
else
{
    // mySearch is private.
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getScriptId()

Gets the script ID of the search. The script ID is available only when the search is either loaded using [nlapiLoadSearch\(type, id\)](#) or has been saved using [nlobjSearch.saveSearch\(title, scriptId\)](#).

If this is an on demand search (created with [nlapiCreateSearch\(type, filters, columns\)](#)), this method will return **null**.

Returns

- The script ID of the search as a string. Typical return values will be something like **customsearch_mysearch** or **customsearchnewinvoices**. You will not receive values such as 55

or 234 or 87. These are considered internal system IDs assigned by NetSuite when you first save the search.

Since

- Version 2012.1

Example

This example shows how to load an existing saved search and get its internal system ID.

```
// Load the existing search and then get its developer-assigned script ID
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
var scriptId = mySearch.getScriptId();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSearchType()

Returns the record type that the search was based on. This method is helpful when you have the internal ID of the search, but do not know the record type the search was based on.

Returns

- The internal ID name of the record type as a string. For example, if the search was on a Customer record, **customer** will be returned; if the search was on the Sales Order record type, **salesorder** will be returned.

Since

- Version 2012.1

Example

```
var searchId = ...;
var s = nlapiLoadSearch(null, searchId); // load a search with an unknown type
var t = s.getSearchType();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

runSearch()

Runs an on demand search, returning the results. Be aware that calling this method does NOT save the search. Using this method in conjunction with [nlapiCreateSearch\(type, filters, columns\)](#) supports creating and running on demand searches that are never saved to the database, much like [nlapiSearchRecord](#).

Note that this method returns the [nlobjSearchResultSet](#) object, which provides you with more flexibility when working with or iterating through your search results. Therefore, you may also want to use **runSearch** in conjunction with **nlapiLoadSearch**. By doing so you can load an existing saved search, call **runSearch**, and then (if you choose):

- retrieve a slice of the search results from anywhere in the result list
- paginate through the search results.

Returns

- [nlobjSearchResultSet](#)

Since

- Version 2012.1

Example 1

This example shows how to load an existing saved search and re-run the search using **runSearch**. After **runSearch** executes, the search's entire result set is returned in an [nlobjSearchResultSet](#) object. You can then use the [forEachResult\(callback\)](#) method to iterate through and process each result.

The callback function receives search result of the search. Remember that the callback function must return **true** or **false**. True causes iteration to continue. False causes iteration to stop.

Note: 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, you must be sure the amount of processing within the callback function does not put the entire script at risk of exceeding scheduled script governance limits.

```
var search = nlapiLoadSearch('opportunity', 'customsearch_kr');
var resultSet = search.runSearch();
var sum = 0;
resultSet.forEachResult(function(searchResult)
{
    sum += parseFloat(searchResult.getValue('total')); // process the search result
    return true; // return true to keep iterating
});
alert('Sum: ' + sum);
```

Example 2

The second example shows another way to define a callback function.

```
// Load a saved search
var search = nlapiLoadSearch('customer', 'customsearch15');

// Run the search to return the results in an nlobjSearchResultSet object
var resultSet = search.runSearch();
// For every result returned, execute the abc() function on the result
resultSet.forEachResult(abc);
/*
 * Define function abc. Function abc is your callback function.
 * This function takes an nlobjSearchResult, and for as long as there is a result returned,
 * call getValue() on the search result column to get the value of the 'fax' column.
 */
function abc(eachResult)
{
    var val = eachResult.getValue('fax');
    return true;
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

saveSearch(title, scriptId)

Saves the search created by [nlobjCreateSearch\(type, filters, columns\)](#).

Executing this API consumes 5 governance units.



Important: Loading a search and saving it with a different title and/or script ID does not create a new search. It only modifies the title and/or script ID for the existing search. To create a new saved search based on an existing search, see Example 2 for [nlobjCreateSearch\(type, filters, columns\)](#).

Parameters

- **title** {string} [optional] - The title you want to give the saved search. Note that *title* is required when saving a new search, but optional when saving a search that was loaded using [nlobjLoadSearch\(type, id\)](#) or has already been saved by calling **saveSearch(title, scriptId)** before.
- **scriptId** {string} [optional] - The script ID you want to assign to the saved search. All saved search script IDs must be prefixed with **customsearch**, for example:

- 'customsearch_my_new_search'
- 'customsearchmynewsearch'

Underscores are not required in your script ID, however, they do improve readability of the script ID.

Also, if you do not provide a script ID for the saved search, the system will generate one for you when the script runs, if the search is being saved for the first time.

Returns

- The internal ID of the search as a number.

Since

- Version 2012.1

Example

This example shows how to create a saved search and assign a title and script ID to the saved search.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', '-5', null );
// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
columns[3] = new nlobjSearchColumn( 'projectedamount' );
columns[4] = new nlobjSearchColumn( 'probability' );
columns[5] = new nlobjSearchColumn( 'email', 'customer' );
columns[6] = new nlobjSearchColumn( 'email', 'salesrep' );
// Create the saved search
var search = nlobjCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch( 'My Opportunities in Last 90 Days', 'customsearch_kr' );
```


[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setColumns(columns)

Sets the return columns for this search, overwriting any prior columns. If **null** is passed in it is treated as if it were an empty array and removes any existing columns on the search.

Parameters

- **columns** {[nlobjSearchColumn\(name, join, summary\)](#)[]} [required] - The [nlobjSearchColumn](#)[] you want to set in the search. Passing in **null** or [] removes all columns from the search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search, load the search, and then redefine the search's search return columns.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'between', 1000, 100000 );
filters[2] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'expectedclosedate' );
columns[2] = new nlobjSearchColumn( 'entity' );
// Create a saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the search
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Define new search columns
var newcolumns = new Array();
newcolumns[0] = new nlobjSearchColumn( 'email' );
newcolumns[1] = new nlobjSearchColumn( 'fax' );
// Override columns from previous search and save new search
mySearch.setColumns(newcolumns);
mySearch.saveSearch('Opportunities email and fax info', 'customsearch_emailfax_kr');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFilterExpression(filterExpression)

Sets the search filter expression, overwriting any prior filters. If **null** is passed in, it is treated as if it was an empty array and removes any existing filters on this search.

Note: This method can be followed by the [addFilter\(filter\)](#) and [addFilters\(filters\)](#) methods. The additional filters will be appended with the current filters on the search through an 'AND' operator.

Parameters

- **filterExpression** {Object[]} [required] - The filter expression you want to set in the search. Passing in null or [] removes all filters from the search.

A search filter expression is a JavaScript string array of zero or more elements. Each element is one of the following:

- Operator - either 'NOT', 'AND', or 'OR'
- Filter term
- Nested search filter expression

For more information about search filter expression, see [Search Filter Expression Overview](#).

Returns

- void

Since

- Version 2012.2

Example

This example shows how to create a saved search, load the search, and then redefine the search filter expression.

```
//Define search filter expression
var filterExpression = [ [ 'trandate', 'onOrAfter', 'daysAgo90' ],
    'or',
    [ 'projectedamount', 'between', 1000, 100000 ],
    'or',
    'not', [ 'customer.salesrep', 'anyOf', -5 ] ];

//Define search columns
var columns = newArray();
columns[0] = new nlobjSearchColumn('salesrep');
columns[1] = new nlobjSearchColumn('entity');

//Create a saved search
var search = nlapiCreateSearch('opportunity', filterExpression, columns);
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');

//Load the search
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');

//Define new search filter expression
var newFilterExpression = [ [ 'customer.salesrep', 'anyOf', -5 ],
    'and',
    [ 'department', , 'anyOf', 3 ] ];

//Override filters from previous search and save new search
```

```
mySearch.setFilterExpression(newFilterExpression);
mySearch.saveSearch('Opportunities salesrep dept', 'customsearch_kr2');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFilters(filters)

Sets the filters for this search, overwriting any prior filters. If **null** is passed in it is treated as if it were an empty array and removes any existing filters on this search.

Note: This method does not accept a search filter expression (**Object[]**) as parameter. Only a search filter list (**nlobjSearchFilter[]**) is accepted. If you want to set a search filter expression, see [setFilterExpression\(filterExpression\)](#).

Parameters

- **filters** {**nlobjSearchFilter[]**} [required] - The **nlobjSearchFilter[]** you want to set in the search. Passing in **null** or **[]** removes all filters from the search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a saved search, load the search, and then redefine the search's filters.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', 'daysAgo90' );
filters[1] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn( 'salesrep' );
columns[1] = new nlobjSearchColumn( 'entity' );
// Create a saved search
var search = nlapiCreateSearch( 'opportunity', filters, columns );
var searchId = search.saveSearch('My Opportunities in Last 90 Days', 'customsearch_kr');
// Load the search
var mySearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
// Define new search filters
var newfilters = new Array();
newfilters[0] = new nlobjSearchFilter( 'salesrep', 'customer', 'anyOf', \-5, null );
newfilters[1] = new nlobjSearchFilter( 'department', null, 'anyOf', 3);

// Override filters from previous search and save new search
mySearch.setFilters(newfilters);
mySearch.saveSearch('Opportunities salesrep dept', 'customsearch_kr2');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setIsPublic(type)

Sets whether the search is public or private. By default, all searches created through [nlapiCreateSearch\(type, filters, columns\)](#) are private.

Parameters

- **type** {boolean} [required] - Set to *true* to designate the search as a public search. Set to **false** to designate the search as a private search.

Returns

- void

Since

- Version 2012.1

Example

This example shows how to create a public saved search.

```
var s = nlapiCreateSearch('Opportunity', filters, columns);
s.setIsPublic(true);
var searchId = s.saveSearch('My public opp search', 'customsearch_opp_public');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setRedirectURLToSearch()

Acts like [nlapiSetRedirectURL\(type, identifier, id, editmode, parameters\)](#) but redirects end users to a populated search definition page. You can use this method with any kind of search that is held in the **nlobjSearch** object. This could be:

- an existing saved search,
- an on demand search that you are building in SuiteScript, or
- a search you have loaded and then modified (using **addFilter**, **setFilters**, **addFilters**, **addColumn**, **addColumns**, or **setColumns**) but do not save.

Note that this method does not return a URL. It works by loading a search into the session, and then redirecting to a URL that loads the search definition page.

This method is supported in afterSubmit user event scripts, Suitelets, and client scripts.

Returns

- void

Since

- Version 2012.1

Example

This example shows that when a user clicks Save in the UI (in an afterSubmit user event script), an existing saved search is loaded into the system. In the UI, the user is taken to the search definition page corresponding to the saved search. The user can then use the UI to redefine the filters or columns for the existing saved search.

```
// Load the search and redirect user to search definition page in the UI
var oppSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
oppSearch.setRedirectURLToSearch();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setRedirectURLToSearchResults()

Acts like [nlapiSetRedirectURL\(type, identifier, id, editmode, parameters\)](#) but redirects end users to a search results page. You can use this method with any kind of search that is held in the nlobjSearch object. This could be:

- an existing saved search,
- an on demand search that you are building in SuiteScript, or
- a search you have loaded and then modified (using **addFilter**, **setFilters**, **addFilters**, **addColumn**, **addColumns**, or **setColumns**) but do not save.

Note that this method does not return a URL. It works by loading a search into the session, and then redirecting to a URL that loads the search results.

This method is supported in afterSubmit user event scripts, Suitelets, and client scripts.

Returns

- void

Since

- Version 2012.1

Example

This example shows that when a user clicks Save in the UI (in an afterSubmit user event script), an existing saved search is loaded into the system. In the UI, the user is taken to the search results page corresponding to the saved search.

```
// Load the search
var oppSearch = nlapiLoadSearch('opportunity', 'customsearch_kr');
oppSearch.setRedirectURLToSearchResults();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

Search Filter Expression Overview

A search filter expression is a JavaScript string array of zero or more elements. Each element is one of the following:

- Operator
- Filter term
- Nested search filter expression

Note: If **any** operator or nested search filter expression is found, then the expression must be **well-formed**. You cannot throw one 'OR' in the middle of three filter terms. You need to have an operator (either 'AND' or 'OR') between each filter term to ensure that expressions are unambiguous and are read properly. Additionally, you are only allowed a maximum depth of three adjacent parentheses, excluding the outermost left and right parentheses. For example: [f1, 'and', [f2, 'and', [f3, 'and', [f4, 'and', f5]]]].

Note: If there are no operators at all and the list contains `nlobjSearchFilter` objects, then the search filter expression is treated as a search filter list. Filters are ANDed together.

Note: Search filter expressions are supported in both client- and server-side scripts.

Operator

An operator (string) can be one of the following:

- 'AND'
- 'OR'
- 'NOT'

The following are the usage guidelines for operators:

- Operators are **case insensitive**. 'and', 'or', and 'not' work the same as 'AND', 'OR', and 'NOT'.
- 'NOT' must be followed by a filter term or a search filter expression.
- 'AND' or 'OR' must be preceded and followed by a filter term or search filter expression.

Filter term

A filter term is a JavaScript array that is composed of three or more elements, as follows:

- Filter identifier - a JavaScript string of the form:
 - **filter_name** (such as `amount`) - This is equivalent to `new nlobjSearchFilter('amount', null, ...)` where 'amount' is the internal ID of the search field.
 - **join_id.filter_name** (such as `customer.salesrep`) - This is equivalent to `new nlobjSearchFilter('salesrep', 'customer', ...)` where 'customer' is the search join id used for the search field specified as filter name 'salesrep'. The filter name in this case may not be a formula filter like "formulatext: ...".
For a list of search join ids and filter names associated to a record, see the [SuiteScript Records Browser](#).
 - **formula_type: formula_text** (such as `formulatext: SUBSTR({custentity_myfield}, 3)`)
 - **aggregate_function(filter_identifier)** (such as `max(amount)`) - The filter_identifier itself can contain a joined record, or can be a formula filter. However, it cannot be both a joined record and a formula filter.
- Operator - a JavaScript string
- Operand - a JavaScript string or integer

- (Optional) Additional operands

nlobjSearchColumn(name, join, summary)

Primary object used to encapsulate search return columns. For information on executing NetSuite searches using SuiteScript, see the help topic [SuiteScript 1.0 Searching Overview](#) in the NetSuite Help Center.

Note: The `columns` argument in `nlapiSearchRecord(type, id, filters, columns)` returns a reference to the `nlobjSearchColumn` object. With the object reference returned, you can then use any of the `nlobjSearchColumn` methods against your search column results.

The `nlobjSearchColumn` object is instantiated with the “new” keyword.

```
var col = new nlobjSearchColumn('email', 'customer');
```

Parameters

- **name** {string} [required] - The search return column name
- **join** {string} [optional] - The join id for this search return column
- **summary** {string} [optional] - The summary type for this column; see the help topic [SuiteScript 1.0 Search Summary Types](#) for additional information
 - group
 - sum
 - count
 - avg
 - min
 - max

Important: If you have multiple search return columns and you apply grouping, all columns must include a summary argument.

In the following example, the first search return column groups the results by tranid. The second search return column returns the count of custbody256 per tranid.

```
filter = new nlobjSearchFilter('type', null, 'is', 'SalesOrd');

var col = new Array();
col[0] = new nlobjSearchColumn('tranid', null, 'group');
col[1] = new nlobjSearchColumn('custbody256', null, 'count');

var result = nlapiSearchRecord('transaction', null, filter, col);
```

nlobjSearchColumn Methods

- [getFormula\(\)](#)
- [getFunction\(\)](#)
- [getJoin\(\)](#)
- [getLabel\(\)](#)

- `getName()`
- `getSort()`
- `getSummary()`
- `setFormula(formula)`
- `setFunction(functionid)`
- `setLabel(label)`
- `setSort(order)`
- `setWhenOrderedBy(name, join)`

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFormula()

Returns

- Returns the formula used for this column as a string

Since

- Version 2009.1

Example

This sample runs a Customer saved search. It uses `getLabel`, `getFormula`, and `getFunction` to return the values specified in the search return columns. In this case of this search, these columns are **Customer Names**, **Customer Names (Reverse)**, **Customer Balance**, and **Phone** (see [Figure 1](#)).

Note that the **Phone** column is a “built-in” column type, so calling `getLabel`, which returns UI label information for custom labels **only**, returns `null`.

```
// reference a Customer saved search
var results = nlapiSearchRecord('customer', 'customsearch81');
var result = results[0];

// return all columns associated with this search
var columns = result.getAllColumns();
var columnLen = columns.length;

// loop through all columns and pull UI labels, formulas, and functions that have
// been specified for columns
for (i = 0; i <= columnLen; i++)
{
    var column = columns[i];
    var label = column.getLabel();
    var formula = column.getFormula();
    var functionName = column.getFunction();
    var value = result.getValue(column);
}
```

To help illustrate the values that `getLabel`, `getFormula`, and `getFunction` are returning, [Figure 1](#) shows the values, as they have been set in the UI, for the formula columns, the column that contains a function, and three of the columns that have custom UI labels.

Figure 1. Figure 1

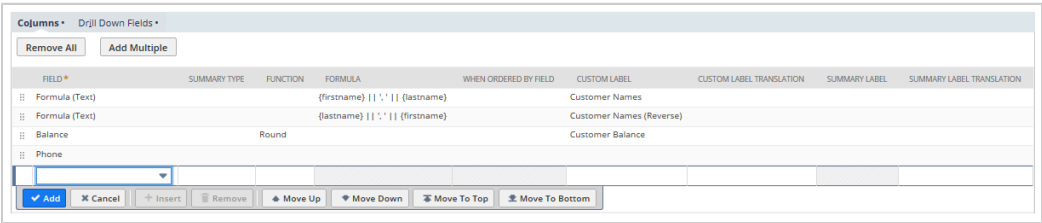


Figure 2 shows the search results after the search is run.

Figure 2. Figure 2

Test Customer Search: Results List Search Audit Trail

[Return To Criteria](#) [Edit this Search](#)

FILTERS

EDIT VIEW INTERNAL ID CUSTOMER NAMES CUSTOMER NAMES (REVERSE) CUSTOMER BALANCE PHONE TOTAL: 31

Edit View	406	Abe, Lincoln	Lincoln, Abe	100.00	789-678-5674
Edit View	385	tasha, brewer	brewer, tasha	0.00	(573) 547-7883
Edit View	335	Angela, Deveroe	Deveroe, Angela	0.00	8642444330
Edit View	218	Brandon, Sommerville	Sommerville, Brandon	60.00	435-950-0598
Edit View	305	Breckenridge, Realty	Realty, Breckenridge	0.00	706-555-9865
Edit View	15	Filippo, Vasta	Vasta, Filippo	0.00	390-75-6978827
Edit View	17	Craig, Koozer	Koozer, Craig	19,382.00	714-541-9852

Figure 3 shows the values for **label** and **formula** for the **Customer Names (Reverse)** column.

Figure 3. Figure 3

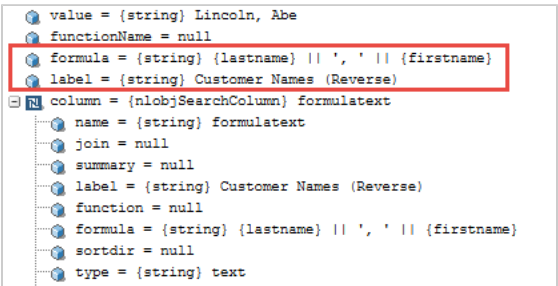
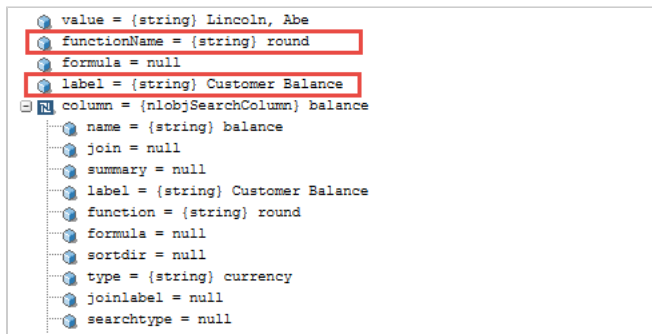


Figure 4 shows the values for **label** and **functionName** for the **Customer Balance** column.

Figure 4. Figure 4

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFunction()

Returns

- The function used in this search column as a string

Since

- Version 2009.1

Example

- See the sample in [getFormula\(\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getJoin()

Returns join id for this search column

Returns

- The join id as a string

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLabel()

Returns the label used for the search column. Note that ONLY custom labels can be returned using this method.

Returns

- The custom label used for this column as a string

Since

- Version 2009.1

Example

- See the sample in [getFormula\(\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getName()

Returns

- The name of the search column as a string

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSort()

Returns the sort direction for this column

Returns

- string

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSummary()

Returns the summary type (avg, group, sum, count) for this search column. In the NetSuite Help Center, see the help topic [SuiteScript 1.0 Search Summary Types](#) for a list of summary types.

Returns

- The summary type as a string

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFormula(formula)

Set the formula used for this column. Name of the column can either be formulatext, formulanumeric, formuladatetime, formulapercent, or formulacurrency.

Parameters

- **formula** {string} [required] - The formula used for this column

Returns

- nlobjSearchColumn

Since

- Version 2011.1

Example

See the example in [Using Formulas, Special Functions, and Sorting in Search](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFunction(functionid)

Sets the special function used for this column.

Parameters

- **functionid** {string} [required] - Special function used for this column. The following is a list of supported functions and their internal IDs:

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
round	Round	No	

ID	Name	Date Function	Output
roundToHundredths	Round to Hundredths	No	
roundToTenths	Round to Tenths	No	
weekOfYear	Week of Year	Yes	text
year	Year	Yes	text

Returns

- nlobjSearchColumn

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Set the label used for this column.

Parameters

- **label** {string} [required] - The label used for this column

Returns

- nlobjSearchColumn

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSort(order)

Returns nlobjSearchColumn sorted in either ascending or descending order.

Parameters

- **order** {boolean} [optional] - If not set, defaults to false, which returns column data in ascending order. If set to true, data is returned in descending order.

Returns

- nlobjSearchColumn

Since

- Version 2010.1

Example 1

Execute a customer search with the customer internal ID in the results. Set the internal ID column to sort in ascending order.

```
var columns = new Array();
columns[0] = new nlobjSearchColumn('internalid');
columns[1] = new nlobjSearchColumn('altname');
columns[2] = columns[0].setSort();
var rec= nlapiSearchRecord('customer', null, null, columns);
```

Example 2

Execute a customer search with the customer internal ID and phone number in the results. Set the results to sort first by phone number and then by internal ID.

```
var columns = new Array();
columns[1] = new nlobjSearchColumn('internalid');
columns[0] = new nlobjSearchColumn('phone');
columns[1].setSort();
columns[0].setSort();
var rec= nlapiSearchRecord('customer', null, null, columns);
```

Example 3

See the example in [Using Formulas, Special Functions, and Sorting in Search](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setWhenOrderedBy(name, join)

Returns the search column for which the minimal or maximal value should be found when returning the **nlobjSearchColumn** value.

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 **nlobjSearchColumn** value for that record is returned.

Can only be used when min or max is passed as the summary parameter in the **nlobjSearchColumn** constructor.

Parameters

- **name** {string} - The name of the search column for which the minimal or maximal value should be found
- **join** {string} - The join id for this search column

Returns

- **nlobjSearchColumn**

Since

- Version 2012.1

Example


Execute a customer search that returns the amount of the most recent sales order per customer.

```
var filters = new Array();
var columns = new Array();
filters[0] = new nlobjSearchFilter("recordtype","transaction","is","salesorder");
filters[1] = new nlobjSearchFilter("mainline","transaction","is","T");
columns[0] = new nlobjSearchColumn("entityid",null,"group");
columns[1] = new nlobjSearchColumn("totalamount","transaction","max");
columns[1].setWhenOrderedBy("trandate","transaction");
var results = nlapiSearchRecord("customer",null,filters,columns);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSearchFilter

Primary object used to encapsulate search filters. For information on executing NetSuite searches using SuiteScript, see the help topic [SuiteScript 1.0 Searching Overview](#) in the NetSuite Help Center.

 **Note:** By default, search filter list (`nlobjSearchFilter[]`) makes use only of an implicit 'AND' operator for filters. This is contrary to search filter expression that can explicitly use either 'AND' or 'OR' operators.

When searching on check box fields, use the **is** operator with a **T** or **F** value to search for checked or unchecked fields, respectively.

To search for a “none of null” value, meaning do not show results without a value for the specified field, use the @NONE@ filter. For example,

```
searchFilters[0] = new nlobjSearchFilter('class', null, 'noneof', '@NONE@');
```

Note that the **filters** argument in `nlapiSearchRecord(type, id, filters, columns)` refers to either a search filter list (`nlobjSearchFilter[]`) or to a search filter expression (`Object[]`). With the object reference returned, you can then use any of the following `nlobjSearchFilter` methods to filter your results.

Methods


- `constructor(name, join, operator, value1, value2)`
- `getFormula()`
- `getJoin()`
- `getName()`
- `getSummaryType()`
- `getOperator()`
- `setFormula(formula)`
- `setSummaryType(type)`

constructor(name, join, operator, value1, value2)

Constructor for a search filter object

Parameters

- **name** {string} - The internal ID of the search field. For example, if one of your filtering criterion is Quantity Available, you will set the value of **name** to **quantityavailable**, which is the search field ID for Quantity Available.
- **join** {string} - If you are executing a joined search, the join id used for the search field specified in the **name** parameter. The join id is the internal ID of the record type the search field appears on.
- **operator** {string} - The search operator used for this search field. For more information about possible operator values, see the help topic [SuiteScript 1.0 Search Operators](#).

 **Note:** If your search filter uses the contains search operator and your search times out, use the haskeywords operator instead.

- **value1** {string | date | string[] | int} - A filter value -or- A special date field value -or- Array of values for select/multiselect fields -or- An integer value
- **value2** {string | date} - A secondary filter value -or- special date field value for between/within style operators * lastbusinessweek. Values are not case sensitive. For more information about possible date filter values, see the help topic [SuiteScript 1.0 Search Date Filters](#).

Returns

- [nlobjSearchFilter](#)

Since

- Version 2007.0

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFormula()

Returns the formula used for this filter

Returns

- The formula used for this filter

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getJoin()

Returns the join id for this search filter

Returns

- The string value of the search join

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getName()

Returns the name for this search filter

Returns

- The string value of the search filter

Since

- Version 2007.0

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSummaryType()

Returns the summary type used for this filter

Returns

- The summary type used for this filter

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getOperator()

Returns the filter operator that was used

Returns

- The string value of the search operator

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFormula(formula)

Sets the formula used for this filter. Name of the filter can either be **formulatext**, **formulanumeric**, **formuladatetime**, **formulapercent**, or **formulacurrency**.

Parameters

- **formula** {string} [required] - The formula used for this filter

Returns

- [nlobjSearchFilter](#)

Since

- Version 2011.1

Example

```
var filters = new Array();
filters[0] = new nlobjSearchFilter('formulatext', null, 'startswith', 'a');
filters[0].setFormula('SUBSTR({custbody_stringfield}, 3)');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

Example

setSummaryType(type)

Sets the summary type used for this filter. Filter name must correspond to a search column if it is to be used as a summary filter.

Parameters

- **type** {string} [required] - The summary type used for this filter. In your script, use one of the following summary type IDs:

Summary type ID (used in script)	Summary Label (as seen in UI)
max	Maximum
min	Minimum
avg	Average (only valid for numeric or currency fields)
sum	Sum (only valid for numeric or currency fields)
count	Count

Returns

- [nlobjSearchFilter](#)

Since

- Version 2011.1

Example

See the sample in [Using Summary Filters in Search](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSearchResult

Primary object used to encapsulate a search result row. For information on executing NetSuite searches using SuiteScript, see the help topic [SuiteScript 1.0 Searching Overview](#) in the NetSuite Help Center.

Methods

- [getAllColumns\(\)](#)
- [getId\(\)](#)
- [getRecordType\(\)](#)
- [getText\(column\)](#)
- [getText\(name, join, summary\)](#)
- [getValue\(name, join, summary\)](#)
- [getValue\(column\)](#)



Note: The following functions return a reference to this object:

- [nlapiSearchDuplicate\(type, fields, id\)](#)
- [nlapiSearchGlobal\(keywords\)](#)
- [nlapiSearchRecord\(type, id, filters, columns\)](#)
- [nlobjSearchResultSet.getResults\(start, end\)](#)

getAllColumns()

Returns an array of [nlobjSearchColumn\(name, join, summary\)](#) objects containing all the columns returned in a specified search

Returns

- [nlobjSearchColumn\[\]](#)

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getId()

Returns the internal ID for the returned record

Returns

- The record internal ID as an integer

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getRecordType()

Returns the record type for the returned record

Returns

- The name of the record type as a string - for example, **customer**, **assemblyitem**, **contact**, or **projecttask**

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getText(column)

Returns the text value for this [nlobjSearchColumn\(name, join, summary\)](#) if it is a select field

Parameters

- **column** {nlobjSearchColumn} [required] - The name of the search result column.

Returns

- string

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getText(name, join, summary)

Returns the UI display name (ie., the text value) for this **nlobjSearchColumn**. Note that this method is supported on **non-stored** select, image, document fields only.

Parameters

- **name** {string} [required] - The name of the search column
- **join** {string} [optional] - The join internalId for this search column
- **summary** {string} [optional] - The summary type used for this search column. Use any of the following types:
 - group
 - sum
 - count
 - avg
 - min
 - max

Returns

- The UI display name for this **nlobjSearchColumn** as a string

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue(name, join, summary)

Returns the value for the **nlobjSearchColumn**

Parameters

- **name** {string} [required] - The name of the search column
- **join** {string} [optional] - The join internalId for this search column
- **summary** {string} [optional] - The summary type used for this search column
 - group
 - sum
 - count
 - avg
 - min
 - max

Returns

- The value for a search return column as a string

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getValue(column)

Can be used on formula fields and non-formula (standard) fields to get the value of a specified search return column

Parameters

- **column** {**nlobjSearchColumn**(name, join, summary)} [required] - Search return column object whose value you want to return

Returns

- String value of the search return column

Since

- Version 2009.1

Example

The following is a Campaign search with joins to the Campaign Recipient record. This sample defines the search return columns, and then uses **getValue()** to return the string value of the email search return column.

```
var filters = new Array();
var columns = new Array();
```

```
// define column objects. See figure for visual representation
columns[0] = new nlobjSearchColumn('title', null, null);
columns[1] = new nlobjSearchColumn('type', 'campaignrecipient', null)
columns[2] = new nlobjSearchColumn('email', 'campaignrecipient', null)

// execute the campaign search
var searchresults = nlapiSearchRecord('campaign', null, filters, columns);

// get the value of the email search return column
var val = searchresults[0].getValue(column
s[2]);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSearchResultSet

Primary object used to encapsulate a set of search results. The **nlobjSearchResultSet** object provides both an iterator interface, which supports processing of each result of the search, and stop at any time, and a slice interface, which supports retrieval of an arbitrary segment of the search results, up to 1000 results at a time.

A **nlobjSearchResultSet** object is returned by a call to **nlobjSearch.runSearch()**, as in:

```
var s = nlapiLoadSearch('opportunity', 'customsearch_cybermonday');
var resultSet = s.runSearch();
```

Methods:

- [forEachResult\(callback\)](#)
- [getColumns\(\)](#)
- [getResults\(start, end\)](#)

forEachResult(callback)

Calls the developer-defined **callback** function for every result in this set. There is a limit of 4000 rows in the result set returned in **forEachResult()**.

Your callback function must have the following signature:

```
boolean callback(nlobjSearchResult result);
```

Note that 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, you must be sure the amount of processing within the callback function does not put the entire script at risk of exceeding scheduled script governance limits.

Also be aware that the execution of the **forEachResult(callback)** method consumes 10 governance units.

Parameters

- **callback** [required] - A JavaScript function. This may be defined as a separate named function, or it may be an anonymous inline function.

Returns

- void

Since

- Version 2012.1

Example

See Example 1 and Example 2 for **nlobjSearch.runSearch()**.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getColumns()

Returns a list of **nlobjSearchColumn** objects for this result set. This list contains one **nlobjSearchColumn** object for each result column in the **nlobjSearchResult** objects returned by this search.

Returns

- **nlobjSearchColumn(name, join, summary)[]**

Since

- Version 2012.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getResults(start, end)

Retrieve a slice of the search result. 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, **getResults(0, 10)** 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 **getResults(20, 30)** will return an array of 5 **nlobjSearchResult** objects.

If more than 1000 rows are required, it is recommended that you modify the search's criteria to reduce the number of results, and the execution time.

Also be aware that the execution of the `getResults(start, end)` method consumes 10 governance units.

Parameters

- **start** {integer} [required] - The index number of the first result to return, inclusive.
- **end** {integer} [required] - The index number of the last result to return, exclusive.

Returns

- `nlobjSearchResult[]`

Throws

- `SSS_INVALID_SEARCH_RESULT_INDEX` if start is negative.
- `SSS_SEARCH_RESULT_LIMIT_EXCEEDED` if more than 1000 rows are requested.

Since

- Version 2012.1

Example

```
// Load a search and get the first three results.
var search = nlapiLoadSearch('opportunity', 'customsearch_al');
var resultSet = search.runSearch();
var firstThreeResults = resultSet.getResults(0, 3);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSelectOption

Primary object used to encapsulate available select options for a select field. This object is returned after a call to `nlobjField.getSelectOptions(filter, filteroperator)`. The object contains a key, value pair that represents a select option, for example: 87, Abe Simpson

Methods:

- `getId()`
- `getText()`

getId()

Use this method to get the internal ID of a select option. For example, on a select field called **Colors**, a call to this method might return 1, 2, 3 (to represent the internal IDs for options that appear in a dropdown field as Red, White, Blue).

Returns

- The integer value of a select option, for example, 1, 2, 3.

Since

- Version 2009.2

Example

```
var myRec = nlapiCreateRecord('opportunity');
myRec.setFieldValue('entity','1');
var myFld = myRec.getField('billaddresslist');
var options = myFld.getSelectOptions('Jones');
nlapiLogExecution('DEBUG', options[0].getId() + ',' + options[0].getText() );
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getText()

Use this method to get the UI display label of a select option. For example, on a select field called **Colors**, a call to this method might return Red, White, Blue.

Returns

- The UI display label of a select option

Since

- Version 2009.2

Example

```
var myRec = nlapiCreateRecord('opportunity');
myRec.setFieldValue('entity','1');
var myFld = myRec.getField('billaddresslist');
var options = myFld.getSelectOptions('Jones');
nlapiLogExecution('DEBUG', options[0].getId() + ',' + options[0].getText() );
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSubrecord

Primary object used to encapsulate a NetSuite subrecord. To create a subrecord, you must first create or load a parent record. You can then create or access a subrecord from a body field or from a sublist field on the parent record.

For general information on subrecords, see the help topic [Working with Subrecords in SuiteScript](#). For a list of all APIs related to subrecords, see [Subrecord APIs](#).

nlobjSubrecord Methods:

- [cancel\(\)](#)
- [commit\(\)](#)

cancel()

Use this method to cancel the current processing of the subrecord and revert subrecord data to the last committed change (submitted in the last `commit()` call).

Note that you will not be able to do any additional write or read operations on the subrecord instance after you have canceled it. You must reload the subrecord from the parent to write any additional data to the subrecord.

Returns

- void

Since

- Version 2011.2

Example

See the help topic [Canceling an Inventory Detail Subrecord](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

commit()

Use this method to commit the subrecord to the parent record. See the help topic [Saving Subrecords Using SuiteScript](#) for additional information on saving subrecords.

Returns

- void

Since

- Version 2011.2

Example

The following sample shows how to use the `commit()` method to commit a subrecord to a parent record. Note that because the subrecord in this script was created from a sublist field, the sublist (the Item sublist in this case), must also be committed to the parent record. Finally, `nlapiSubmitRecord()` is called on the parent to commit all changes to the database.

```
var record = nlapiCreateRecord('purchaseorder', {recordmode: 'dynamic'});
record.setFieldValue('entity', 38);
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'quantity', 1);
record.setCurrentLineItemValue('item', 'item', 108);

//create new subrecord from the Inventory Details field on the Items sublist
var subrecord = record.createCurrentLineItemSubrecord('item', 'inventorydetail');
subrecord.selectNewLineItem('inventoryassignment');
subrecord.setCurrentLineItemValue('inventoryassignment', 'issueinventorynumber', 'testinv2343');
subrecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 1);
subrecord.commitLineItem('inventoryassignment');
//commit Inventory Detail subrecord to parent record
```

```

subrecord.commit();

//commit changes to the Items sublist to the parent record
record.commitLineItem('item');

//commit parent record
var id =nlapiSubmitRecord(record);

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

UI Objects

SuiteScript UI objects are a collection of objects that can be used as a UI toolkit for server scripts such as Suitelets and user event scripts. UI objects encapsulate scriptable user interface components such as NetSuite portlets, forms, fields, lists, sublists, tabs, and columns. They can also encapsulate all components necessary for building a custom NetSuite-looking assistant wizard. If you are not familiar with UI objects, see the help topic [UI Objects Overview](#).

UI Objects:

- [nlobjAssistant](#)
- [nlobjAssistantStep](#)
- [nlobjButton](#)
- [nlobjColumn](#)
- [nlobjField](#)
- [nlobjFieldGroup](#)
- [nlobjForm](#)
- [nlobjList](#)
- [nlobjPortlet](#)
- [nlobjSubList](#)
- [nlobjTab](#)
- [nlobjTemplateRenderer](#)

Important Things to Note:

- When you add a UI object to an **existing** NetSuite page, the internal ID used to reference the object must be prefixed with **custpage**. This minimizes the occurrence of field/object name conflicts. See the help topic [SuiteScript 1.0 Creating Custom NetSuite Pages with UI Objects](#) for more information.
- Although UI objects give developers a lot of control over the characteristics, placement, and behaviors of UI elements, developer resources need to be spent creating and maintaining them. During design time, application architects should carefully weigh the trade off between customizing the NetSuite UI with SuiteBuilder, versus programmatically customizing it with SuiteScript UI objects. (For information about working with SuiteBuilder point-and-click customization tools, see the help topic [SuiteBuilder Overview](#) in the NetSuite Help Center.)

nlobjAssistant

Primary object used to encapsulate all properties of a scriptable multi-step NetSuite assistant. All data and state for an assistant is tracked automatically throughout the user's session up until completion of the assistant.

For examples showing how to build and run an assistant in your NetSuite account, see the help topic [Building a NetSuite Assistant with UI Objects](#).

Methods

- `addField(name, type, label, source, group)`
- `addFieldGroup(name, label)`
- `addStep(name, label)`
- `addSubList(name, type, label)`
- `getAllFields()`
- `getAllFieldGroups()`
- `getAllSteps()`
- `getAllSubLists()`
- `getCurrentStep()`
- `getField(name)`
- `getFieldGroup(name)`
- `getLastAction()`
- `getLastStep()`
- `getNextStep()`
- `getStep(name)`
- `getStepCount()`
- `getSubList(name)`
- `hasError()`
- `isFinished()`
- `sendRedirect(response)`
- `setCurrentStep(step)`
- `setError(html)`
- `setFieldValues(values)`
- `setFinished(html)`
- `setNumbered(hasStepNumber)`
- `setOrdered(ordered)`
- `setScript(script)`
- `setShortcut(show)`
- `setSplash(title, text1, text2)`
- `setTitle(title)`

`addField(name, type, label, source, group)`

Use this method to add a field to an assistant and return the field object.

Parameters

- **name** {string} [required] - The internal ID for this field
- **type** {string} [required] - The field type. Any of the following field types can be specified:
 - `text`

- email
- radio - See [Working with Radio Buttons](#) for details on this field type.
- label - This is a field type that has no values. In [Working with Radio Buttons](#), see the first code sample that shows how to set this field type.
- phone
- date
- currency
- float
- integer
- checkbox
- select - Note that if you want to add your own (custom) options on a select field, you must set the *source* parameter to NULL. Then, when a value is specified, the value will populate the options from the source.
- url - See the help topic [Create a Form with a URL Field](#) for an example how to use this type.
- timeofday
- textarea
- multiselect
- image
- inlinehtml
- password
- help
- percent
- longtext



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.

- richtext
- **label** {string} [optional] - The UI label for this field
- **source** {int | string} [optional] - The internalId or scriptId of the source list for this field if it is a select (List/Record) field. In the NetSuite Help Center, see [List/Record Type IDs](#) for the internal IDs of all supported list/record types.



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with `nlobjField.addSelectOption(value, text, selected)`. The select values are determined by the source record or list.

Note that if you have set the **type** parameter to **select**, and you want to add your own (custom) options to the select field, you must set **source** to NULL. Then, when a value is specified, the value will populate the options from the source.

- **group** {string} [optional] - If you are adding the field to a field group, specify the internal ID of the field group

Returns

- [nlobjField](#)

Since

- Version 2009.2

Example

This snippet shows the addition of a field group to an assistant object. In the UI, the field group appears as the Company Information group. Two text fields (**Company Name** and **Legal Name**) are added to the Company Information field group. Help text is added to the Legal Name field.

```
assistant.addFieldGroup("companyinfo", "Company Information")
assistant.addField("companyname", "text", "Company Name", null, "companyinfo");
assistant.addField("legalname", "text", "Legal Name", null, "companyinfo");

assistant.getField("legalname").setHelpText("Enter a Legal Name if it differs from your company name")
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addFieldGroup(name, label)

Use this method to add a field group to an assistant page. Note that when a field group is added to an assistant, by default it is collapsible. Also, by default, it will appear as uncollapsed when the page loads. If you want to change these behaviors, you will use the `nlobjFieldGroup.setCollapsible(collapsible, hidden)` method.

Parameters

- name** {string} [required] - The internal ID for the field group
- label** {string} [required] - The UI label for the field group

Returns

- `nlobjFieldGroup`

Since

- Version 2009.2

Example 1

This snippet shows how to add a field group called Company Info an assistant page. It also shows how to add fields to the field group. Finally, the `nlobjAssistant.getField(name)` method is used to return the **legalname** and **shiptoattention** field objects. After these fields are returned, help text is added to each of these fields.

```
assistant.addFieldGroup("companyinfo", "Company Information")
    .setHelpText("Setup your <b>important</b> company information in the fields below.");
assistant.addField("companyname", "text", "Company Name", null, "companyinfo");
assistant.addField("legalname", "text", "Legal Name", null, "companyinfo");
assistant.addField("shiptoattention", "text", "Ship To Attention", null, "companyinfo");
assistant.addField("address1", "text", "Address 1", null, "companyinfo").setLayoutType("normal", "startcol");
assistant.addField("address2", "text", "Address 2", null, "companyinfo");
assistant.addField("city", "text", "City", null, "companyinfo");
```

```
assistant.getField("legalname").setHelpText("Enter a Legal Name if it differs from your company name");
assistant.getField("shiptoattention").setHelpText("Enter the name of someone
who can sign for packages or important documents. This is important because
otherwise many package carriers will not deliver to your corporate address");
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addStep(name, label)

Use this method to add a step to an assistant.

Parameters

- **name** {string} [required] - The internal ID for this step (for example, 'entercontacts').
- **label** {string} [required] - The UI label for the step (for example, 'Enter Contacts'). By default, the step will appear vertically in the left panel of the assistant (see figure).

Note: You can position your steps horizontally (directly below the title of the assistant) by setting `nlobjAssistant.setOrdered(ordered)` to **false**. Note that if you do this, users will be able to complete steps in a random order.

Returns

- `nlobjAssistantStep`

Since

- Version 2009.2

Example 1

This snippet shows how to add a step to the left panel. Steps must include an internal ID and a UI label. After the step is added, a `nlobjAssistantStep` object is returned. Through this object you can use `setHelpText(help)` if you want to create help text for the step.

```
assistant.addStep('companyinformation', 'Setup Company Information').setHelpText("Setup your <b>important</b>  
company information in the fields below.");
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSubList(name, type, label)

Use this method to add a sublist to an assistant page and return an [nlobjSubList](#) object. Note that only inlineeditor sublists can be added to assistant pages.

Parameters

- **name** {string} [required] - The internal ID for the sublist
- **type** {string} [required] - The sublist type. Currently, only a value of **inlineeditor** can be set.
- **label** {string} [required] - The UI label for the sublist

Returns

- [nlobjSubList](#)

Since

- Version 2009.2

Example

This snippet shows that when a user navigates to a step that has the internal ID **entercontacts**, a sublist called Contacts is added to the page. Notice the use of the [nlobjSubList.setUniqueField\(name\)](#) method in this example. This method is used to define the **Name** field as a unique field in the sublist. This means that when users enter values into this field, the values must be unique. In other words, users cannot enter two instances of Sally Struthers in the Name field.

```
else if (step.getName() == "entercontacts")
{
    var sublist = assistant.addSubList("contacts", "inlineeditor", "Contacts")
    sublist.addField("name", "text", "Name");
}
```



```

sublist.addField("phone", "phone", "Phone");
sublist.addField("email", "email", "E-mail");
sublist.addField("address", "textarea", "Address");
sublist.setUniqueField("name");
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllFields()

Use this method to get all fields in an assistant. Regardless of which page or step the fields have been added to, all fields will be returned. Also note that where you call this method matters. If you call `getAllFields()` early in your script, and then add ten more fields at the end of your script, `getAllFields()` will return only those fields that were added prior to the call.

Returns

- String[] of all fields in a custom assistant

Since

- Version 2009.2

Example

See Example 2 for `getField(name)`. Also see the help topic [UI Object Assistant Code Sample](#), which shows how to use `getAllFields()` within the context of an assistant workflow.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllFieldGroups()

Use this method to get all field groups on an assistant page. Also note that where you call this method matters. If you call `getAllFieldGroups()` early in your script, and then add three more field groups at the end of your script, `getAllFieldGroups()` will return only those field groups that were added prior to the call.

Returns

- String[] of all field groups in the assistant

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllSteps()

Use this method to return an array of all the assistant steps for this assistant.

Returns

- [nlobjAssistantStep\[\]](#)

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllSubLists()

Use this method to get all sublist names that appear on an assistant page. Also note that where you call this method matters. If you call `getAllSubLists()` early in your script, and then add three more sublists at the end of your script, `getAllSubLists()` will return only those sublists that were added prior to the call.

Returns

- `String[]` of all sublists in an assistant

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getCurrentStep()

Use this method to get the current step that was set via `nlobjAssistant.setCurrentStep(step)`. After getting the current step, you can add fields, field groups, and sublists to the step.

Returns

- [nlobjAssistantStep](#)

Since

- Version 2009.2

Example

For examples that show how to use `getCurrentStep()` within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

getField(name)

Use this method to return a field on an assistant page.

Parameters

- **name** {string} [required] - The internal ID of the field

Returns

- [nlobjField](#)

Since

- Version 2009.2

Example 1

This snippet shows how to add a text field called **Legal Name**. The field is being added to a field group with the internal ID **companyinfo**. After the field has been added, an **nlobjField** object is returned. The **getField(name)** method is then used to get the field object and set help text. The help text appears directly below the field.

```
assistant.addField("legalname", "text", "Legal Name", null, "companyinfo");
assistant.getField("legalname").setHelpText("Enter a Legal Name if it differs from your company name")
```

Example 2

This snippet shows how to use **getField(name)** for something other than adding help text to a field. In the case, **getField(name)** is used in conjunction with the **nlobjAssistant.getAllFields()** method. After all field objects in the assistant are returned, the **getField(name)** method is used to loop through each field so that values can be set for the fields.

```
var fields = assistant.getAllFields()
for (var i = 0; i < fields.length; i++)
{
  assistant.getField(fields[i]).setDefaultValue(nlapiGetContext().getSessionObject(fields[i]))
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldGroup(name)

Use this method to return a field group on an assistant page.

Parameters

- **name** {string} [required] - The internal ID for the field group

Returns

- [nlobjFieldGroup](#)

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLastAction()

Use this method to get the last submitted action that was performed by the user. Typically you will use [getNextStep\(\)](#) to determine the next step (based on the last action).

Possible assistant submit actions that can be specified are:

- **next** - means that the user has clicked the Next button in the assistant
- **back** - means that the user has clicked the Back button
- **cancel** - means that the user has clicked the Cancel button
- **finish** - means that the user has clicked the Finish button. By default, inline text then appears on the finish page saying "Congratulations! You have completed the <assistant title>" - where <assistant title> is the title set in [nlobjCreateAssistant\(title, hideHeader\)](#) or [nlobjAssistant.setTitle\(title\)](#).
- **jump** - if [nlobjAssistant.setOrdered\(ordered\)](#) has been set to false (meaning that steps can be completed in random order), then **jump** is used to get the user's last action in a non-sequential process.

Returns

- The last submit action (as a string)

Since

- Version 2009.2

Example

For examples that show how to use `getLastAction()` within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLastStep()

Use this method to get the step the last submitted action came from.

Returns

- [nlobjAssistantStep](#)

Since

- Version 2009.2

Example

For examples that show how to use `getLastStep()` within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getNextStep()

Use this method to return the next logical step corresponding to the user's last submitted action. You should only call this method after you have successfully captured all the information from the last step and are ready to move on to the next step. You would use the return value to set the current step prior to continuing.

Returns

- `{nlobjAssistantStep}` Returns the next logical step based on the user's last submit action, assuming there were no errors. Typically you will call `setCurrentStep(step)` using the returned step if the submit was successful.

Since

- Version 2009.2

Example

For examples that show how to use `getNextStep()` within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getStep(name)

Use this method to return an `nlobjAssistantStep` object on an assistant page.

Parameters

- **name** {string} [required] - The internal ID of the step

Returns

- `nlobjAssistantStep`

Since

- Version 2009.2

Example 1

This sample shows how to create a step and then set the step as the current step in the assistant.

```
//create a step that has an internal ID of 'companyinformation'
assistant.addStep('companyinformation', 'Setup Company Information');

// later in the script, set the current step to the step identified as companyinformation
assistant.setCurrentStep(assistant.getStep('companyinformation'));
```

Example 2

For examples that show how to use `getStep()` within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getStepCount()

Use this method to get the total number of steps in an assistant.

Returns

- The total number of steps in an assistant. Value returned as an integer.

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSubList(name)

Use this method to return a sublist on an assistant page .

Parameters

- **name** {string} [required] - The internal ID for the sublist

Returns

- [nlobjSubList](#)

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

hasError()

Use this method to determine if an assistant has an error message to display for the current step.

Returns

- Returns true if [setError\(html\)](#) was called

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

isFinished()

Use this method to determine when all steps in an assistant are completed.

Returns

- Returns true if all steps in the assistant have been completed or if [setFinished\(html\)](#) has been called.

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

sendRedirect(response)

Use this method to manage redirects in an assistant. In most cases, an assistant redirects to itself as in:

```
nlobjSetRedirectURL('suitelet', nlobjGetContext().getScriptId(), nlobjGetContext().getDeploymentId());
```

The **sendRedirect(response)** method is like a wrapper method that performs this redirect. 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 the user Finishes. This method, when used in the second assistant, ensures that the user is redirected back to the first assistant.

Parameters

- **response** {[nlobjResponse](#)} [required] - The response object

Returns

- void

Since

- Version 2009.2

Example

For examples that show how to use **sendRedirect(response)** within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setCurrentStep(step)

Use this method to mark a step as the current step. In the UI, the step will be highlighted when the user is on that step (see figure).

Parameters

- **step** {[nlobjAssistantStep](#)} [required] - The name of the step object

Returns

- void

Since

- Version 2009.2

Example 1

This snippet sets the user's current step to the **companyinformation** step. Notice the step is automatically highlighted in the left panel.

```
assistant.setCurrentStep(assistant.getStep("companyinformation"));
```

Example 2

For examples that show how to use **setCurrentStep(step)** within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setError(html)

Use this method to set an error message for the current step. If you choose, you can use HTML tags to format the message.

Parameters

- **html** {string} [required] - Error message text

Returns

- void

Since

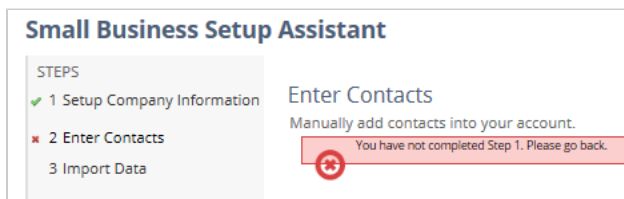
- Version 2009.2

Example

This snippet shows how to use **setError(html)** to display an error message on a step.

```
else if (step.getName() == "entercontacts")
{
  assistant.setError("You have not completed Step 1. Please go back.");
  var sublist = assistant.addSubList("contacts", "inlineeditor", "Contacts")
  sublist.addField("name", "text", "Name");

  // remainder of code...
}
```



[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValues(values)

Use this method to set values for fields on an assistant page.

Parameters

- **values** {hashtable<string, string>} [required] - An associative array containing name/value pairs that map field names to field values

Returns

- void

Since

- Version 2009.2

Example

This snippet shows how to add two text fields to an assistant, and then programmatically set the value of each field.

```
assistant.addField("companyname", "text", "Company Name", null, "companyinfo");
assistant.addField("address1", "text", "Address 1", null, "companyinfo")
assistant.setFieldValues({companyname: "Wolfe Electronics", address1: "123 Main St., Anytown, USA"});
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFinished(html)

Use this method to mark the last page in an assistant. Set the rich text to display a completion message on the last page.

Parameters

- **html** {string} [required] - The text to display when the assistant has finished. For example, "Congratulations! You have successfully set up your account."

Returns

- void

Since

- Version 2009.2

Example

For examples that show how to use **setFinished(html)** within the context of an assistant workflow, see the help topic [UI Object Assistant Code Sample](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setNumbered(hasStepNumber)

Use this method to display steps without numbers.

Parameters

- **hasStepNumber** {boolean} [optional] - Set to false to turn step numbering off.

Returns

- void

Since

- Version 2010.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setOrdered(ordered)

Use this method to enforce a sequential ordering of assistant steps. If steps are ordered, users must complete the current step before the assistant will allow them to proceed to the next step. From a display

perspective, ordered steps will always appear in the left panel of the assistant (see first figure). Note that by default, steps in an assistant are ordered.

If steps are unordered, they can be completed in any order. Additionally, unordered steps are always displayed horizontally under the assistant title (see second figure).

Parameters

- **ordered** {boolean} [required] - A value of **true** means steps must be completed sequentially, and that they will appear vertically in the left panel of the assistant. A value of **false** means steps do not need to be completed sequentially, and they will appear horizontally, directly below the assistant title.

ordered parameter set to **true**:

The screenshot shows the 'Small Business Setup Assistant' interface. On the left, a vertical sidebar titled 'STEPS' contains three items: '1 Setup Company Information' (highlighted with a dark background), '2 Enter Contacts', and '3 Import Data'. The main content area is titled 'Setup Company Information' and includes the instruction 'Setup your **important** company information in the fields below.' Below this, there are input fields for 'SHIP TO ATTENTION', 'ADDRESS 1', 'ADDRESS 2', 'CITY', 'COMPANY NAME' (with a blue border), and 'LEGAL NAME'. A dropdown arrow is visible next to the 'Company Information' section header.

ordered parameter set to **false** :

The screenshot shows the 'Small Business Setup Assistant' interface with a horizontal progress bar at the top. The progress bar has three steps: '1 Setup Company Information' (active, with a blue circle), '2 Enter Contacts', and '3 Import Data'. Below the progress bar, the main content area is titled 'Setup Company Information' and includes the instruction 'Setup your **important** company information in the fields below.' The input fields for 'SHIP TO ATTENTION', 'ADDRESS 1', 'ADDRESS 2', 'CITY', 'COMPANY NAME' (with a blue border), and 'LEGAL NAME' are arranged in a more spread-out layout compared to the first figure. A dropdown arrow is visible next to the 'Company Information' section header.

Returns

- void

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setScript(script)

Use this method to set the scriptId for a global client script you want to run on an assistant page.

Parameters

- **script** {string | int} [required] - The scriptId of the global client script

Returns

- void


Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setShortcut(show)

Use this method to show/hide the **Add to Shortcuts** link that appears in the top-right corner of an assistant page. Note that if you do not call this method in your script, the default is to show the Add to Shortcuts link at the top of all assistant pages. Therefore, it is recommended that you use this method only if you want to hide this link.

 **Note:** The Add to Shortcuts link is always hidden on external pages.

Parameters

- **show** {boolean} [required] - A value of false means that the Add to Shortcuts link does not appear on the assistant. A value of true means that it will appear.

Returns

- void

Since

- Version 2009.2

Example

This snippet shows that with **setShortcut(show)** set to false, the Add to Shortcuts link will not display on assistant pages.

```
var assistant = nlapiCreateAssistant("Small Business Setup Assistant", true);
assistant.setOrdered(true);
assistant.setShortcut(false);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSplash(title, text1, text2)

Use this method to set the splash screen for an assistant page.

Parameters

- **title** {string} [required] - The title of the splash screen
- **text1** {string} [required] - Text for the splash screen
- **text2** {string} [optional] - If you want splash content to have a two-column appearance, provide content in the text2 parameter.

Returns

- void

Since

- Version 2009.2

Example

The following figure show a splash page that appears when `setSplash()` is set. Note the two-column layout in this example. The second column appears because text has been passed to the text2 parameter.

```
assistant.setCurrentStep(assistant.getStep("companyinformation"));
assistant.setSplash("Welcome to the Small Business Setup Assistant!",
  "<b>What you'll be doing</b><br>The Small Business Setup Assistant will
  walk you through the process of configuring your NetSuite account for
  your initial use..", "<b>When you finish</b><br>your account will be ready
  for you to use to run your business.");
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setTitle(title)

Use this method to set the title for the assistant. If you have already defined the title using `nlapiCreateAssistant(title, hideHeader)`, you do not need to call the `setTitle(title)` method. Also note that the title you provide using `setTitle(title)` will override the title specified in the `nlapiCreateAssistant()` function.

Parameters

- **title** {string} [required] - Assistant title

Returns

- void

Since

- Version 2009.2

Example

This sample shows that if you set the title using `setTitle(title)`, you will override the title specified in `nlapicreateAssistant()`.

```
function showAssistant(request, response)
{
    /* first create assistant object and define its steps. */
    var assistant = nlapicreateAssistant("Small Business Setup Assistant");
    assistant.setTitle("Small Business Setup Assistant");

    //remainder of code ....
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjAssistantStep

Primary object used to encapsulate a step within a custom NetSuite assistant.

For information on working with `nlobjAssistantStep` objects, as well as information on building an assistant using other UI objects, see the help topic [Building a NetSuite Assistant with UI Objects](#).

Methods

- [getAllFields\(\)](#)
- [getAllLineItemFields\(group\)](#)
- [getAllLineItems\(\)](#)
- [getFieldValue\(name\)](#)
- [getFieldValues\(name\)](#)
- [getLineItemCount\(group\)](#)
- [getLineItemValue\(group, name, line\)](#)
- [getStepNumber\(\)](#)
- [setHelpText\(help\)](#)
- [setLabel\(label\)](#)

getAllFields()

Use this method to get all fields entered by the user during the step.

Returns

- String[] of all fields entered by the user during the step

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllLineItemFields(group)

Use this method to get all sublist fields entered by the user during this step.

Parameters

- **group** {string} [required]- The sublist internal ID

Returns

- String[] of all sublist fields entered by the user during the step

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getAllLineItems()

Use this method to get all sublists entered by the user during this step.

Returns

- String[] of all sublists entered by the user during this step

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValue(name)

Use this method to get the value of a field entered by the user during this step.

Parameters

- **name** {string} [required] - The internal ID of the field whose value is being returned

Returns

- The internal ID (string) value for the field

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getFieldValues(name)

Use this method to get the selected values of a multi-select field as an Array.

Parameters

- **name** {string} [required]- The name of the multi-select field

Returns

- String[] of field IDs. Returns **null** if field is not on the record. Note the values returned are **read-only**.

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemCount(group)

Use the method to get the number of lines previously entered by the user in this step.



Important: The first line number on a sublist is **1** (not 0).

Parameters

- **group** {string} [required]- The sublist internal ID

Returns

- The integer value of the number of line items on a sublist. Note that -1 is returned if the sublist does not exist.

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemValue(group, name, line)

Use this method to get the value of a line item (sublist) field entered by the user during this step.

Parameters

- **group** {string} [required] - The sublist internal ID
- **name** {string} [required]- The name of the sublist field whose value is being returned
- **linenum** {int} [required]- The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of the sublist field

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getStepNumber()

Use this method to get a step number. The number returned represents where this step appears sequentially in the assistant.

Returns

- The index of this step in the assistant page (1-based)

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHelpText(help)

Use this method to set help text for an assistant step.

Parameters

- **help** {string} [required] - The help text for the step

Returns

- nlobjAssistantSte

Since

- Version 2009.2

Example

See the sample provided in `nlobjAssistant.addStep(name, label)`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Use this method to set the label for an assistant step. Note that you can also create a label for a step when the step is first added to the assistant. Do this using `nlobjAssistant.addStep(name, label)`.

Parameters

- **label** {string} [required] - The UI label for this step

Returns

- `nlobjAssistantStep`

Since

- Version 2009.2

Example

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjButton

Primary object used to encapsulate custom buttons. Note that custom buttons only appear in the UI when the record is in Edit mode. Custom buttons do not appear in View mode. Also note that in SuiteScript, buttons are typically added to a record or form in **beforeLoad** user event scripts.



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](#) in the NetSuite Help Center for details.

Methods

- [setDisabled\(disabled\)](#)
- [setLabel\(label\)](#)
- [setVisible\(visible\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDisabled(disabled)

Disables the button. When using this API, the assumption is that you have already defined the button's UI label when you created the button using `nlobjForm.addButton(name, label, script)`. The `setDisabled()` method grays-out the button's appearance in the UI.



Important: This method is not currently supported for standard NetSuite buttons. This method can be used with custom buttons only.

Parameters

- **disabled** {boolean} - If set to **true**, the button will still appear on the form, however, the button label will be grayed-out.

Returns

- nlobjButton

Since

- Version 2008.2

Example

```
function disableUpdateOrderButton(type, form)
{
    //Get the button
    var button = form.getButton('custpage_updateorder');

    //Disable the button in the UI
    button.setDisabled(true);
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Sets the UI label for the button. When using this API, the assumption is that you have already defined the button's UI label when you created the button using **nlobjForm.addButton(name, label, script)**. You can set **setLabel()** to trigger based on the execution context. For example, based on the user viewing a page, you can use **setLabel()** to re-label a button's UI label so that the label is meaningful to that particular user.

This API is supported on standard NetSuite buttons as well as on custom buttons. For a list of standard buttons that support this API, see the help topic [Button IDs](#) in the NetSuite Help Center.

Parameters

- **label** {string} - The UI label for the custom button

Returns

- nlobjButton

Since

- Version 2008.2

Example

```
function relabelUpdateOrderButton(type, form)
{
    //Get the button
    var button = form.getButton('custpage_updateorderbutton');

    //Relabel the button's UI label
    button.setLabel('Modify Order');
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setVisible(visible)

Sets the button as hidden in the UI. This API is supported on custom buttons and on *some* standard NetSuite buttons. For a list of standard buttons that support this API, see the help topic [Button IDs](#) in the NetSuite Help Center.

Parameters

- **visible** {boolean} - Defaults to true if not set. If set to false, the button will be hidden in the UI.

Returns

- nlobjButton

Since

- Version 2010.2

Example

```
function hideSaveAndPrintButton(type, form)
{
    //Get the button
    var button = form.getButton('saveprint');

    //Make sure that the button is not null
    if(button != null)
        //Hide the button in the UI
        button.setVisible(false);
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjColumn

Primary object used to encapsulate list columns. To add a column, you must first create a custom list using [nlapiCreateList\(title, hideNavbar\)](#), which returns an [nlobjList](#) object.

After the list object is instantiated, you can add a standard column using the [nlobjList.addColumn\(name, type, label, align\)](#) method.

You can also add an "Edit | View" column using the [nlobjList.addEditColumn\(column, showView, showHrefCol\)](#) method. Both methods return an [nlobjColumn](#) object.

nlobjColumn Methods

- [addParamToURL\(param, value, dynamic\)](#)
- [setLabel\(label\)](#)
- [setURL\(url, dynamic\)](#)

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addParamToURL(param, value, dynamic)

Adds a URL parameter (optionally defined per row) to this column's URL. Should only be called after calling [setURL\(url, dynamic\)](#)

Parameters

- **param** {string} [required] - The parameter name added to the URL
- **value** {string} [required] - The parameter value added to the URL - or - a column in the data source that returns the parameter value for each row
- **dynamic** {boolean} [optional] - If true, then the parameter value is actually an alias that is calculated per row

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Sets the UI label for this column

Parameters

- **label** {string} [required] - The UI label used for this column

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setURL(url, dynamic)

Sets the base URL (optionally defined per row) for this column

Parameters

- **url** {string} [required] - The base URL or a column in the data source that returns the base URL for each row

- **dynamic** {boolean} [optional] - If true, then the URL is actually an alias that is calculated per row

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjField

Primary object used to encapsulate a NetSuite field.

Important Things to Note about nlobjField:

- To add a **nlobjField** object to an existing NetSuite form (that appears on a record), use a `beforeLoad` user event script. See the help topic [Enhancing NetSuite Forms with User Event Scripts](#) for an example.
- To add a **nlobjField** object to a Suitelet, you must create a custom form using [nlapiCreateForm\(title, hideNavbar\)](#), which returns an **nlobjForm** object. After the form object is instantiated, add a new field to the form using the **nlobjForm.addField(name, type, label, sourceOrRadio, tab)** method, which returns a reference to **nlobjField**.
- To return a reference to an **nlobjField** object, use [nlapiGetField\(fldnam\)](#) (for body fields) or [nlapiGetLineItemField\(type, fldnam, linenum\)](#) (for sublist fields). If you do not know the difference between a body field and a sublist field, see the help topic [Working with Fields Overview](#) in the NetSuite Help Center.
- If you use [nlapiGetField\(fldnam\)](#) in a **client script** to return a **nlobjField** object, the object returned is **read-only**. This means that you can use **nlobjField** getter methods on the object, however, you cannot use **nlobjField** setter methods to set field properties. There is one exception, **nlobjField.setDisplayType** can be used to set the display type.
- Be aware of any special permissions that might be applied to a field. For example, a permission error will be thrown if you attempt to get select options on a field that has been disabled on a form.

Methods

- [addSelectOption\(value, text, selected\)](#)
- [getLabel\(\)](#)
- [getName\(\)](#)
- [getSelectOptions\(filter, filteroperator\)](#)
- [getType\(\)](#)
- [setAlias\(alias\)](#)
- [setBreakType\(breaktype\)](#)
- [setDefaultValue\(value\)](#)
- [setDisplaySize\(width, height\)](#)
- [setDisplayType\(type\)](#)
- [setHelpText\(help, inline\)](#)

- `setLabel(label)`
- `setLayoutType(type, breaktype)`
- `setLinkText(text)`
- `setMandatory(mandatory)`
- `setMaxLength(maxlength)`
- `setPadding(padding)`
- `setRichTextHeight(height)`
- `setRichTextWidth(width)`

addSelectOption(value, text, selected)

Adds a select option to a SELECT field



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with `nlobjField.addSelectOption`. The select values are determined by the source record or list.

Parameters

- **value** {string} [required] - The internal ID of this select option
- **text** {string} [required] - The UI label for this option
- **selected** {boolean} [optional] - If true, then this option is selected by default

Returns

- void

Since

- Version 2008.2

Example

This snippet shows how to add a select field to a form. Use `addSelectOption()` to define the options that will be available to this field.

```
// add a select field and then add the select options that will appear in the dropdown list
var select = form.addField('selectfield', 'select', 'My Custom Select Field');
select.addSelectOption("", "");
select.addSelectOption('a', 'Albert');
select.addSelectOption('b', 'Baron');
select.addSelectOption('c', 'Chris');
select.addSelectOption('d', 'Drake');
select.addSelectOption('e', 'Edgar');
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLabel()

Returns field UI label

Returns

- String value of the field's UI label

Since

- Version 2009.1

Example

```
function getFieldInfo(type, form)
{
    var field = nlapiGetField('memo'); // specify internalId of Memo field on a Sales Order
    alert(field.getType()); // returns text as the field type for memo
    alert(field.getLabel()); // returns Memo as the field UI label
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getName()

Returns the field internal ID

Returns

- String value of a field's internal ID

Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSelectOptions(filter, filteroperator)


Use this API to obtain a list of available options on a select field. This API can be used on both standard and custom select fields. Only the first 1,000 available options will be returned by this API.

This method can only be used in server contexts against a record object. Also note that a call to this method may return different results for the same field for different roles.

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.

Parameters

- 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.

 **Note:** Filter values are case insensitive. The filters 'John' and 'john' will return the same select options.

- filteroperator** {string} [optional] - Supported operators are **contains** | **is** | **startswith**. If not specified, defaults to the **contains** operator.

Returns

- An array of [nlobjSelectOption](#) objects. These objects represent the key-value pairs representing a select option (for example: **87, Abe Simpson**).

Since

- Version 2009.2

Example 1

This sample shows how to get a filtered set of select options available to the Customer (entity) field on an Opportunity record. Only the select options that start with the letter **C** will be returned.

```
var myRec = nlapiLoadRecord('opportunity', 333);
var myField = myRec.getField('entity');
var options = myField.getSelectOptions('C', 'startswith');
```

Example 2

This sample shows how to create a Sales Order record and then set the Customer (entity) field to a specific customer (87). Based on the customer specified, the script then gets available select options on the **Bill To Select (billaddresslist)** field.

```
var myRec = nlapiCreateRecord('salesorder');
myRec.setFieldValue('entity', '87');
var myFld = myRec.getField('billaddresslist');
var options = myFld.getSelectOptions();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getType()

Returns the field type - for example, *text*, *date*, *currency*, *select*, *checkbox*, etc.

Returns

- String value of field's SuiteScript type

Since

- Version 2009.1

Example

```
function getFieldInfo(type, form)
{
    var field = nlapiGetField('memo'); // specify internalId of Memo field on a Sales Order
    alert(field.getType()); // returns text as the field type for memo
    alert(field.getLabel()); // returns Memo as the field UI label
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setAlias(alias)

Sets the alias used to set the value for this field. By default the alias is equal to the field's name. The method is only supported on scripted fields via the UI Object API.

Parameter:

- **alias** {string} [required] - The value used to override the alias

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setBreakType(breaktype)

Use this method to set the layout type for a field and optionally the break type. This method is only supported on scripted fields that have been created using the UI Object API.

Parameter:

- **breaktype** {string} [required] - The break type used to add a break in flow layout for this field. Available types are:
 - **startcol** - This starts a new column (also disables automatic field balancing if set for any field)
 - **startrow** - For outside fields, this places the field on a new row. The startrow breaktype is only used for fields with a layout type of outside. See [setLayoutType\(type, breaktype\)](#).
 - **none** - (default)

Returns

- [nlobjField](#)

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDefaultValue(value)

Sets the default value for this field. This method is only supported on scripted fields via the UI object API.

Parameters

- **value** {string} [required] - The default value for this field. Note that if you pass an empty string, the field will default to a blank field in the UI.

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDisplaySize(width, height)

Sets the height and width for the field. Only supported on multi-selects, long text, rich text, and fields that get rendered as INPUT (type=text) fields. This API is not supported on list/record fields. This method is only supported on scripted fields via the UI object API.

Parameters

- **width** {int} [required]- The width of the field (cols for textarea, characters for all others)
- **height** {int} [optional]- The height of the field (rows for textarea and multiselect fields)

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDisplayType(type)

Sets the display type for this field.

Be aware that this method cannot be used in **client** scripts. In other words, if you use [nlapiGetField\(fldnam\)](#) in a client script to return a field object that has been added to a form, you cannot use **setDisplayType** to set the field's display type. The **nlobjField** object returned from [nlapiGetField\(fldnam\)](#) is **read-only**.

Parameters

- **type** {string} [required] - The display type for this field.
 - **inline** - This makes the field display as inline text
 - **hidden** - This hides the field on the form.
 - **readonly** - This disables the field but it is still selectable and scrollable (for textarea fields)
 - **entry** - This makes the sublist field appear as a data entry input field (for non checkbox, select fields)
 - **disabled** - This disables the field from user-changes
 - **normal** - (default) This makes the field appear as a normal input field (for non-sublist fields)



Important: If the display type of a field is set to **hidden** (with either SuiteBuilder or SuiteScript), you must set the display type to **normal** to make the field appear.

Returns

- [nlobjField](#)

Since

- Version 2008.2

Example

This sample shows a user event script, which specifies the **hidden** parameter to hide a check box field on a beforeLoad event. When the record is loaded (for example, an Estimate or Customer record), the check box referenced in this script will be hidden from the user.

```
function beforeLoad(type, form)
{
    form.getField('custbody_myspecialcheckbox').setDisplayType('hidden');
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHelpText(help, inline)

Use this method to set help text for this field.

Parameters

- **help** {string} [required]- Help text for the field. When the field label is clicked, a field help popup will open to display the help text.
- **inline** {boolean} [optional]- If not set, defaults to false. This means that field help will appear **only** in a field help popup box when the field label is clicked. If set to true, field help will display in a field help popup box, as well as inline below the field (see figure).



Important: The **inline** parameter is available **only** to **nlobjField** objects that have been added to [nlobjAssistant](#) objects. The *inline* parameter is not available to fields that appear on [nlobjForm](#) objects.

Returns

- [nlobjField](#)

Since

- Version 2009.2

Example

The following snippet shows how to use the **getField(name)** method to get a field object that has been added to an assistant. Then **setHelpText(help, inline)** is used to add field help. The help will appear in a field help popup when the field is clicked. Because the **inline** parameter has been set to true, the field help will also appear inline, directly below the field.

```
assistant.getField("legalname").setHelpText("Enter a Legal Name if it differs from your  
company name", true);
```

The screenshot shows the 'Small Business Setup Assistant' window. On the left, a 'STEPS' sidebar lists '1 Setup Company Information', '2 Enter Contacts', and '3 Import Data'. The main area is titled 'Setup Company Information' and contains instructions: 'Setup your **important** company information in the fields below.' Below this are input fields for 'SHIP TO ATTENTION', 'ADDRESS 1', 'ADDRESS 2', and 'CITY'. A section titled 'Company Information' contains 'COMPANY NAME' and 'LEGAL NAME' fields. A 'Field Help' dialog box is open over the 'LEGAL NAME' field, displaying the text: 'Enter a Legal Name if it differs from your company name' and 'Field ID: legalname'. At the bottom right of the main window are 'Cancel', '< Back', and 'Next >' buttons.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Sets the UI label for this field. The method is available only on scripted fields via the UI object API.

Parameters

- **label** {string} [required]- The UI label used for this field

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLayoutType(type, breaktype)

Sets the display type for this field and optionally the break type. This method is only supported on scripted fields via the UI Object API.

Parameters

- **type** {string} [required] - The layout type for this field. Use any of the following layout types:

- **outside** - This makes the field appear outside (above or below based on form default) the normal field layout area
- **outsidebelow** - This makes the field appear below the normal field layout area
- **outsideabove** - This makes the field appear above the normal field layout area
- **startrow** - This makes the field appear first in a horizontally aligned field group in normal field layout flow
- **midrow** - This makes the field appear in the middle of a horizontally aligned field group in normal field layout flow
- **endrow** - This makes the field appear last in a horizontally aligned field group in normal field layout flow
- **normal** - (default)
- **breaktype** {string} [required] - The layout break type. Use any of the following break types.
 - **startcol** - This starts a new column (also disables automatic field balancing if set for any field)
 - **startrow** - For outside fields, this places the field on a new row
 - **none** - (default)

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLinkText(text)

Sets the text that gets displayed in lieu of the field value for URL fields.

Parameters

- **text** {string} [required] - The displayed value (in lieu of URL)

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMandatory(mandatory)

Sets the field to mandatory. The method is only supported on scripted fields via the UI Object API.

Parameters

- **mandatory** {boolean} [required]- If true, then the field will be defined as mandatory

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setMaxLength(maxlength)

Sets the max length for this field (only valid for text, rich text, long text, and textarea fields). This method is only supported on scripted fields via the UI Object API.

Parameters

- **maxLength** {int} [required]- The max length for this field

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setPadding(padding)

Sets the number of empty field spaces before/above this field. This method is only supported on scripted fields via the UI Object API.

Parameters

- **padding** {int} [required] - The number of empty vertical spaces (rows) before this field

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setRichTextHeight(height)

If **Rich Text Editing** is enabled, you can use this method to set the height of the rich text field only. You can set a separate height for the text area using [setDisplaySize\(width, height\)](#). When setting the height, the minimum value is 100 pixels and the maximum value is 500 pixels.

For information on enabling the Rich Text Editor, see the help topic [Personal Preferences for Appearance](#).

Parameters

- **height** {int} [optional]- The height of the field (pixels).

Returns

- [nlobjField](#)

Since

- Version 2015.1

Example

```
function demoSimpleForm(request,response) {
    var form = nlapiCreateForm('Simple Form');

    var field = form.addField('custpage_richtext','richtext','Rich Text', null,null);

    field.setDisplaySize(200,50);
    field.setRichTextWidth(500);
    field.setRichTextHeight(200);

    response.writePage( form );
}
```

setRichTextWidth(width)

If **Rich Text Editing** is enabled, you can use this method to set the width of the rich text field only. You can set a separate width of the text area using [setDisplaySize\(width, height\)](#). When setting the width, the minimum value is 250 pixels and the maximum value is 800 pixels.

For information on enabling the Rich Text Editor, see the help topic [Personal Preferences for Appearance](#)

Parameters

- **width** {int} [optional]- The width of the field (pixels).

Returns

- [nlobjField](#)

Since

- Version 2015.1

Example

See the example for [setRichTextHeight\(height\)](#).

nlobjFieldGroup

Primary object used to encapsulate a field group on a custom NetSuite assistant page and on [nlobjForm](#) objects.

You can create an assistant by calling `nlapicreateAssistant(title, hideHeader)`, which returns a reference to the `nlobjAssistant` object. On the assistant object, call `addFieldGroup` to instantiate a new `nlobjFieldGroup` object.


To learn more about field groups, see the help topic [Building a NetSuite Assistant with UI Objects](#).

Methods

- `setCollapsible(collapsible, hidden)`
- `setLabel(label)`
- `setShowBorder(show)`
- `setSingleColumn(column)`

setCollapsible(collapsible, hidden)

Use this method to define whether a field group can be collapsed. You can also use this method to define if the field group will display as collapsed or expanded when the page first loads.

 **Note:** This method is not currently supported on field groups that have been added to `nlobjForm` objects. This method can only be used on field groups added on `nlobjAssistant` objects.

Parameters

- **collapsible** {boolean} [required] - A value of true means that the field group can be collapsed. A value of false means that the field group cannot be collapsed - the field group displays as a static group that cannot be opened or closed.
- **hidden** {boolean} [optional] - If not set, defaults to false. This means that when the page loads, the field group will not appear collapsed. Note: If you set the collapsible parameter to false (meaning the field group is not collapsible), then any value you specify for **hidden** will be ignored.

Returns

- `nlobjFieldGroup`

Since

- Version 2009.2

Examples

The following figure shows three field groups.

Field group 1 has been set to:

```
assistant.addFieldGroup("companyprefs", "Company Preferences").setCollapsible(true, false);
```

This means that the field group **is** collapsible, and that when the page loads, the field group will display as **uncollapsed**. Note that this is the default appearance of a field group. If you add a field group and do not call `setCollapsible`, the field group will appear as it does in field group 1 in the figure below.

Field group 2 has been set to:

```
assistant.addFieldGroup("accountingprefs", "Accounting Preferences").setCollapsible(true, true);
```

This means that the field group **is** collapsible, and that when the page loads, the field group will display as **collapsed**.

Field group 3 has been set to:

```
assistant.addFieldGroup("accountingprefsmore", "Even More Accounting Preferences").setCollapsible(false);
```

This means that the field group **is not** collapsible. Notice that field group 3 does not contain the triangle icon that controls collapsibility.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Use this method to create a UI label for a field group.

Parameters

- **label** {string} [required] - The UI label for the field group

Returns

- nlobjFieldGroup

Since

- Version 2009.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setShowBorder(show)

Use this method to conditionally show or hide the border of a field group. A field group border consists of the field group title and the gray line that frames the group by default.

Parameters

- **show** {boolean} [required] - Set to true to show a field group border. Set to false to hide the border.

Returns

- void

Since

- Version 2011.1

Example

See the sample for `nlobjForm.addFieldGroup(name, label, tab)`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSingleColumn(column)

Use this method to determine how your field group is aligned. You can choose to align it into a single column or allow NetSuite to auto-align it.

Parameters

- **column** {boolean} [required] - Set to true to place all fields in the field group into a single column. Set to false to allow NetSuite to auto-align your field group fields into one, two, or three columns, depending on the number of fields and the width of your screen.

Returns

- void

Since

- Version 2011.1

Example

See the sample for `nlobjForm.addFieldGroup(name, label, tab)`.

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjForm

Primary object used to encapsulate a NetSuite-looking form. Note that the `nlapiCreateForm(title, hideNavbar)` function returns a reference to this object.

Methods

- `addButton(name, label, script)`
- `addCredentialField(id, label, website, scriptId, value, entityMatch, tab)`
- `addField(name, type, label, sourceOrRadio, tab)`

- `addFieldGroup(name, label, tab)`
- `addPageLink(type, title, url)`
- `addResetButton(label)`
- `addSubList(name, type, label, tab)`
- `addSubmitButton(label)`
- `addSubTab(name, label, tab)`
- `addTab(name, label)`
- `getButton(name)`
- `getField(name, radio)`
- `getSubList(name)`
- `getSubTab(name)`
- `getTab(name)`
- `getTabs()`
- `insertField(field, nextfld)`
- `insertSubList(sublist, nextsub)`
- `insertSubTab(subtab, nextsub)`
- `insertTab(tab, nexttab)`
- `removeButton(name)`
- `setFieldValues(values)`
- `setScript(script)`
- `setTitle(title)`

addButton(name, label, script)

Adds a button to a form

Parameters

- **name** {string} [required] - The internal ID name of the button. The internal ID must be in lowercase, contain no spaces, and include the prefix **custpage** 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 **custpage_updateorder**.
- **label** {string} [required] - The UI label used for this button
- **script** {string} [optional]- The onclick script used for this button

Returns

- `nlobjButton`

Since

- Version 2008.2

Example:

```
function SimpleFormWithButton(request, response)
```

```

{
  if ( request.getMethod() == 'GET' )
  {
    var form = nlapiCreateForm('Simple Form with Button');

    var script = "alert('Hello World')";

    form.addButton('custombutton', 'Click Me', script);

    response.writePage( form );
  }
  else
    dumpResponse(request,response);
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addCredentialField(id, label, website, scriptId, value, entityMatch, tab)

Adds a field that lets you store credentials in NetSuite to be used when invoking services provided by third parties. For example, merchants need to store credentials in NetSuite used to communicate with Payment Gateway providers when executing credit card transactions.

This method is supported in client and server scripts.

Additional things to note about this method:

- 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).



Important: The default maximum length for a credential field is 32 characters. If needed, use [setMaxLength\(maxlength\)](#) to change this value.

Parameters

- **id** {string} [required] - The internal ID of the credential field.
- **label** {string} [required] - The UI label for the credential field.
- **website** {string} [optional] - The domain the credentials can be sent to. For example, 'www.mysite.com'. This value can also be an array of strings representing a list of domains to which the credentials can be sent using [nlapiRequestUrlWithCredentials](#). Note that although no exception is thrown if this parameter value is not passed, [nlapiRequestURLWithCredentials\(credentials, url, postdata, headers, httpsMethod\)](#) will not work without it.
- **scriptId** {string} [optional] - The scriptId of the script that is allowed to use this credential field. For example, 'customscript_my_script'. Note that although no exception is thrown if this parameter value is not passed, [nlapiRequestURLWithCredentials\(credentials, url, postdata, headers, httpsMethod\)](#) will not work without it.

- **value** {string} [required] - If you choose, you can set an initial value for this field. This value is the handle to the credentials. If you do not want to set a value, you must pass in a null value or empty string.
- **entityMatch** {boolean} [optional] - Controls whether use of `nlapirequesturlwithcredentials` with this credential is restricted to the same entity that originally entered the credential. An example where you would not want this (you would set to **false**) is with a credit card processor, where the credential represents the company an employee is working for and multiple entities will be expected to make secure calls out to the processor (clerks, for example). An example where you might want to set **entityMatch** to **true** is when each user of the remote call has his or her own credentials.
- **tab** {string} [optional] - The tab parameter can be used to specify either a tab or a field group (if you have added `nlobjFieldGroup` objects to your form). If tab is empty, then the field is added to the "main" section of the form.

Returns

- `nlobjField` object

Since

- Version 2012.1

Example

This sample shows how to create a form and add a credential field to the form. In the UI, the credential field will appear to users as Username. After the user submits the form, the credentials will be sent to a website called `www.mysite.com`. Additionally, only a script with the script ID 'customscript_a' can use this credential field. Finally, the **entityMatch** parameter is set to **false**. This means that when `nlapirequesturlwithcredentials(credentials, url, postdata, headers, httpsMethod)` is used with this credential there is no restriction to the same entity that originally entered the credential.

```
function demoSimpleForm(request, response)
{
    if (request.getMethod() == "GET")
    {
        var form = nlapiCreateForm('Simple Form');
        var credField = form.addCredentialField('username', 'Username',
            'www.mysite.com', 'customscript_a', null, false, null);
        credField.setMaxLength(64);
        form.addSubmitButton('Save');
        response.writePage( form );
    }
}
```

This code shows how to retrieve the credential in a Suitelet.

```
function demoSimpleForm(request, response)
{
    if (request.getMethod() == "POST")
    {
        var handle = request.getParameter("username");
    }
}
```

This code shows how to use the credential.

```
var creds= [record.getFieldValue("username")];
var sUrl = "https://www.mysite.com/serviceA?user=" + "{" + record.getFieldValue("username") + "}";

response = nlapiRequestURLWithCredentials(creds, sUrl, null, null, null);
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addField(name, type, label, sourceOrRadio, tab)

Adds an [nlobjField](#) object to a form and returns a reference to it

Parameters

- **name** {string} [required] - The internal ID name of the field. The internal ID must be in lowercase, contain no spaces, and include the prefix **custpage** if you are adding the field to an existing page. For example, if you add a field that appears as **Purchase Details**, the field internal ID should be something similar to **custpage_purchasedetails** or **custpage_purchase_details**.
- **type** {string} [required] - The field type for this field. Use any of the following enumerated field types:
 - text
 - radio - See [Working with Radio Buttons](#) for details on adding this field type.
 - label - This is a field type that has no values. It is used for placing a label next to another field. In [Working with Radio Buttons](#), see the first code sample that shows how to set this field type and how it will render in the UI.
 - email
 - phone
 - date
 - datetimetz - This field type lets you combine date and time values in one field. For example, you may want a single field to contain date and time “timestamp” data. After a user enters a date/time value, the data is rendered in the user's preferred date and time format, as well as the user's time zone. Also note that time values are stored in NetSuite down to the second.
 - currency
 - float
 - integer
 - checkbox
 - select
 - url - See the help topic [Create a Form with a URL Field](#) for an example how to use this type.
 - timeofday
 - textarea
 - multiselect
 - image - This field type is available **only** for fields appearing on list/staticlist sublists. You cannot specify an *image* field on a form.
 - inlinehtml
 - password
 - help
 - percent

- `longtext`



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.

- `richtext`
- `file` - This field type is available only for Suitelets and will appear on the main tab of the Suitelet page. File upload fields cannot be added to tabs, subtabs, sublists, or field groups and are not allowed on existing pages. Setting the field type to **file** adds a file upload widget to the page and changes the form encoding type for the form to multipart/form-data. See [Uploading Files to the File Cabinet Using SuiteScript](#) for an example of creating a **file** field type, and then later retrieving this file using the `nlobjRequest.getFile(id)` method.
- **label** {string} [optional] - The UI label for this field (this is the value displayed for help fields)
- **source** {int | string} [optional] - The internalId or scriptId of the source list for this field if it is a select (List/Record) or multi-select field. See [List/Record Type IDs](#) for the internal IDs of all supported list/record types.



Important: If you are adding a field of type 'radio', the value of the **source** parameter will be the radio button's unique ID. See the first code sample in [Working with Radio Buttons](#) for details.



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with `nlobjField.addSelectOption(value, text, selected)`. The select values are determined by the source record or list.

- **tab** {string} [optional]- The **tab** parameter can be used to specify either a tab or a field group (if you have added `nlobjFieldGroup` objects to your form). If **tab** is empty, then the field is added to the "main" section of the form.



Note: If any of the following field types is set to hidden, the object returned is text.

- Checkbox
- Radio
- Select
- Textarea

Returns

- `nlobjField`

Since

- Version 2008.2

Example

This samples shows how to create a new form using `nlapiCreateForm`, add two tabs to the form using `nlobjForm.addTab`, and then add one field to each new tab using `nlobjForm.addField`.

```
//Create a form called Interns.
var newForm = nlapiCreateForm('Interns');

//Add a tab to the Intern form.
```



```

var firstTab = newForm.addTab('custpage_academichistorytab', 'Academic History');

//Add a text field to the first tab.
newForm.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Intern form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
newForm.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');

```

The screenshot shows a web form titled 'Interns'. It has two tabs: 'Academic History' and 'Student Contact'. The 'Student Contact' tab is selected. Below the tabs, there is a label 'STUDENT EMAIL' and a text input field containing the email address 'kwolfe@ucdavis.com'.

List/Record Type IDs

If you are adding a **select** (List/Record) field, refer to the following IDs when providing a value for the **source** parameter. You can use the IDs specified in either the Internal ID or the Internal ID (number) columns. Many people reference the Internal ID because it is easier to remember and makes more sense in the context of their code.

Note: When referencing a custom record as the **source**, use the record's custom scriptid. This will prevent naming conflicts should you later share your script using SuiteBundler capabilities. (For information on bundling scripts, see the help topic [SuiteBundler Overview](#) in the NetSuite Help Center).

Record Type	Internal ID	Internal ID (number)
Account	account	-112
Accounting Period	accountingperiod	-105
Call	phonecall	-22
Campaign	campaign	-24
Campaign Event	campaignevent	-107
Case	supportcase	-23
Class	classification	-101
Competitor	competitor	-108
Contact	contact	-6
Currency	currency	-122
Customer	customer	-2
Customer Category	customercategory	-109
Department	department	-102
Email Template	emailtemplate	-120

Record Type	Internal ID	Internal ID (number)
Employee	employee	-4
Employee Type	employeetype	-111
Entity Status	customerstatus	-104
Event	calendarevent	-20
Field	customfield	-124
Issue	issue	-26
Item	item	-10
Item Type	itemtype	-106
Location	location	-103
Module	issuemodule	-116
Opportunity	opportunity	-31
Partner	partner	-5
Product	issueproduct	-115
Product Build	issueproductbuild	-114
Product Version	issueproductversion	-113
Project	job	-7
Project Task	projecttask	-27
Promotion Code	promotioncode	-121
Record Type		-123
Role		-118
Saved Search		-119
Scripted Record Type		-125
Solution	solution	-25
Subsidiary	subsidiary	-117
Task	task	-21
Transaction	transaction	-30
Transaction Type	transactiontype	-100
Vendor	vendor	-3
Vendor Category	vendorcategory	-110

Working with Radio Buttons

Through SuiteScript you can add fields of type 'radio' to both [nlobjForm](#) and [nlobjAssistant](#) objects. The 'radio' field type is unique in that if you add a series of radio fields, the **name** parameter in

`nlobjForm.addField(name, type, label, sourceOrRadio, tab)` must be the same for each field. (Typically, the value of **name** is unique among all fields in a script.)

Fields of type 'radio' are differentiated by the values set in the **source** parameter. For radio fields only, the **source** parameter contains the internal ID for the field.

See these sections for more information:

- [Adding Radio Fields](#)
- [Getting Radio Fields](#)
- [Setting Radio Fields](#)
- [Getting Radio Options](#)

Adding Radio Fields

The following sample shows two sets of radio buttons added to a Suitelet. One set is called 'orgtype', which will display vertically on the form; the second set is called 'companysize', which will display horizontally. The **name** parameter for each set is the same, whereas the value of **source** is different for all radio fields.

```
function radioButtonSamples(request, response)
{
    var form = nlapiCreateForm('Sample Form');

    // create a field of type 'label' - this field type holds no data and is used for display purposes only
    form.addField('orgtypelabel', 'label', 'What type of organization are you?').setLayoutType('startrow');

    /* add fields of type 'radio'. Notice that this set of radio buttons all specify 'orgtype' as the field
    * name for each button. Each radio button is distinguished by the value specified in
    * the 'source' parameter. By default, this set of radio fields will appear vertically since
    * no layout type has been specified
    */
    form.addField('orgtype', 'radio', 'Business To Consumer', 'b2c');
    form.addField('orgtype', 'radio', 'Business To Business', 'b2b');
    form.addField('orgtype', 'radio', 'Non-Profit', 'nonprofit');

    //default the "Business to Business" radio button as selected when the page loads
    form.getField('orgtype', 'b2b').setDefaultValues(['b2b']);

    /* now add the second set of radio buttons. Notice that this group also shares the same
    * value for name, which is 'companysize'. Also note the use of the setLayoutType method.
    * Use this when you want to position the buttons horizontally.
    */
    form.addField('companysizelabel', 'label', 'How big is your organization?').setLayoutType('startrow');
    form.addField('companysize', 'radio', 'Small (0-99 employees)', 's').setLayoutType('midrow');
    form.addField('companysize', 'radio', 'Medium (100-999 employees)', 'm').setLayoutType('midrow');
    form.addField('companysize', 'radio', 'Large (1000+ employees)', 'l').setLayoutType('endrow');

    response.writePage(form);
}
```

Sample Form

WHAT TYPE OF ORGANIZATION ARE YOU? HOW BIG IS YOUR ORGANIZATION? ☐ SMALL (0-99 EMPLOYEES) ☐ MEDIUM (100-999 EMPLOYEES) ☐ LARGE (1000+ EMPLOYEES)

☐ BUSINESS TO CONSUMER

☒ BUSINESS TO BUSINESS

☐ NON-PROFIT

Notice that if you set the radio button using `setLayoutType('midrow')`, the radio button appears to the left of the text. If `setLayoutType` is not used, the buttons will appear to the right of the text.

Getting Radio Fields

The following snippet shows how to get values for a radio button using `nlobjForm.getField()`. Note that you must specify the radio button name, as well as its source parameter, which represents its internal ID.

```
form.assistant.getField('orgtype', 'b2c');
```

Setting Radio Fields

You can set the value of a radio button using either of these approaches:

```
form.getField('orgtype', 'b2c').setDefaultValues(['b2c']);
```

- or -

```
form.setFieldValues({ orgtype : 'b2c' });
```

In either case, when the page loads the radio button with the ID 'b2c' (Business To Consumer) will appear as selected.

Getting Radio Options

This sample shows that you can also use `nlobjField.getSelectOptions(filter, filteroperator)` on radio buttons as well as on select fields. When this method is used on radio buttons, you will get an array of `nlobjSelectOption` values representing each radio button.

```
form.addField('orgtype', 'radio', 'Business To Consumer', 'b2c').setLayoutType('endrow');
form.addField('orgtype', 'radio', 'Business To Business', 'b2b').setLayoutType('midrow');
form.addField('orgtype', 'radio', 'Non-Profit', 'nonprofit').setLayoutType('endrow');

var orgtypeoptions = form.getField('orgtype').getSelectOptions();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addFieldGroup(name, label, tab)

Adds a field group to the form.

Parameters

- **name** {string} [required] - Provide an internal ID for the field group.
- **label** {string} [required] - The UI label for the field group
- **tab** {string} [optional] - Specify the tab you the field group to appear on. If no tab is specified, the field group is placed on the "main" area of the form.

Returns

- `nlobjFieldGroup`

Since

- Version 2011.1

Example

```
function formWithFieldGroups(request, response)
{
    if ( request.getMethod() == 'GET' )
    {
        var form = nlapiCreateForm('Simple Form');
        var group = form.addFieldGroup( 'myfieldgroup', 'My Field Group');
        form.addField('companyname', 'text', 'Company Name', null, 'myfieldgroup');
        form.addField('legalname', 'text', 'Legal Name', null, 'myfieldgroup');
        form.addField('datefield', 'date', 'Date', null, 'myfieldgroup' );
        form.addField('currencyfield', 'currency', 'Currency', null, 'myfieldgroup');
        form.addField('textareafield', 'textarea', 'Textarea', null, 'myfieldgroup');
        group.setShowBorder(true);

        //Add a tab to the form.
        var firstTab = form.addTab('academichistorytab', 'Academic History');

        //Add a second tab to the form.
        var secondTab = form.addTab('studentcontacttab', 'Student Contact');

        //Add a field group to the first tab and align all field group fields in a single column
        var fieldGroupUniv = form.addFieldGroup("universityinfo", "Univeristy Information",
            'academichistorytab');
        fieldGroupUniv.setSingleColumn(true);

        //Add fields to the University Information field group.
        form.addField('universityname', 'text', 'University Name', null, "universityinfo");
        form.addField('universityaddr', 'text', 'University Address', null, "universityinfo");

        //Add a field group to the second tab.
        form.addFieldGroup('studentinfogroup', "Student Information", 'studentcontacttab');

        //Add fields to the Student Information field group.
        form.addField('studentemail', 'email', 'Student Email', null, "studentcontacttab");
        form.addField('studentphone1', 'text', 'Student Phone 1', null, "studentcontacttab");
        form.addField('studentphone2', 'text', 'Student Phone 2', null, "studentcontacttab");
        form.addField('studentphone3', 'text', 'Student Phone 3', null, "studentcontacttab");
        form.addField('studentphone4', 'text', 'Student Phone 4', null, "studentcontacttab");

        form.addSubmitButton('Submit');
        response.writePage( form );
    }
    else
        dumpResponse(request, response);
}
```

```
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addPageLink(type, title, url)

Adds a navigation cross-link to the form

Parameters

- **type** {string} [required] - The type of navbar link to add. Possible values include:
 - **breadcrumb** - appears on top left corner after system bread crumbs
 - **crosslink** - appears on top right corner
- **title** {string} [required] - The text displayed in the link
- **url** {string} [required] - The URL used for this link

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addResetButton(label)

Adds a reset button to a form

Parameters

- **label** {string} [optional]- The UI label used for this button. If no label is provided, the label defaults to **Reset**.

Returns

- [nlobjButton](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSubList(name, type, label, tab)

Adds an [nlobjSubList](#) object to a form and returns a reference to it. Note that sorting (in the UI) is not supported on static sublists created using the **addSubList()** method if the row count exceeds 25.

Parameters

- **name** {string} [required] - The internal ID name of the sublist. The internal ID must be in lowercase, contain no spaces, and include the prefix **custpage** if you are adding the sublist to an existing page. For example, if you add a sublist that appears on the UI as *Purchase Details*, the sublist internal ID should be something equivalent to **custpage_purchasedetails** or **custpage_purchase_details**.
- **type** {string} [required] - The sublist type. Use any of the following types:
 - **editor** - An edit sublist with non-inline form fields (similar to the Address sublist)
 - **inlineeditor** - An edit sublist with inline fields (similar to the Item sublist)
 - **list** - A list sublist with editable fields (similar to the Billable Items sublist)
 - **staticlist** - A read-only segmentable list sublist (similar to the search results sublist)
- **label** {string} [required] - The UI label for this sublist
- **tab** {string} [optional] - The tab under which to display this sublist. If empty, the sublist is added to the main tab.

Returns

- [nlobjSubList](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSubmitButton(label)

Adds a submit button to a form

Parameters

- **label** {string} [optional] - The UI label for this button. If no label is provided, the label defaults to **Save**.

Returns

- [nlobjButton](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSubTab(name, label, tab)

Adds a subtab to a form and returns an [nlobjTab](#) object reference to it.



Important: If you add only one subtab, the UI label you define for the subtab will not appear in the UI. You must define two subtabs for subtab UI labels to appear.

Parameters

- **name** {string} [required] - The internal ID name of the subtab. The internal ID must be in lowercase, contain no spaces, and include the prefix **custpage** if you are adding the subtab to an existing page. For example, if you add a subtab that appears on the UI as **Purchase Details**, the subtab internal ID should be something similar to **custpage_purchasedetails** or **custpage_purchase_details**.
- **label** {string} [required] - The UI label of the subtab
- **tab** {string} [optional] - The tab under which to display this subtab. If empty, it is added to the main tab.

Returns

- [nlobjTab](#)

Since

- Version 2008.2

Example

See the sample in the section [Adding Subtabs with SuiteScript](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addTab(name, label)

Adds a tab to a form and returns an [nlobjTab](#) object reference to the tab

Parameters

- **name** {string} [required] - The internal ID name of the tab. The internal ID must be in lowercase, contain no spaces, and include the prefix **custpage** if you are adding the tab to an existing page. For example, if you add a tab that appears on the UI as **Purchase Details**, the tab internal ID should be something equivalent to **custpage_purchasedetails** or **custpage_purchase_details**.
- **label** {string} [required] - The UI label of the tab

Returns

- [nlobjTab](#)

Since

- Version 2008.2

Example

This sample shows how to create a new form using [nlapiCreateForm\(title, hideNavbar\)](#), add two tabs to the form using [nlobjForm.addTab\(name, label\)](#), and then add one field to each new tab using [nlobjForm.addField](#).

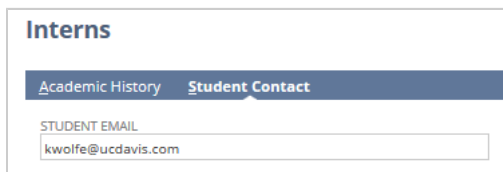
```
//Create a form called Interns.
var newForm = nlapiCreateForm('Interns');

//Add a tab to the Intern form.
var firstTab = newForm.addTab('custpage_academichistorytab', 'Academic History');

//Add a text field to the first tab.
newForm.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Intern form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
newForm.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');
```



[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getButton(name)

Returns an [nlobjButton](#) object by name

Parameters

- **name** {string} [required] - The internal ID of the button. Internal IDs must be in lowercase and contain no spaces.

Returns

- [nlobjButton](#)

Since

- Version 2008.2

Example

```
function disableUpdateOrderButton(type, form)
{
    //Get the button before relabeling or disabling
    var button = form.getButton('custpage_updateorderbutton');

    //Disable the button in the UI
    button.setDisabled(true);
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getField(name, radio)

Returns an [nlobjField](#) object by name

Parameters

- **name** {string} [required] - The internal ID name of the field. Internal ID names must be in lowercase and contain no spaces.
- **radio** {string} - If this is a radio field, specify which radio field to return based on the radio value.

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSubList(name)

Returns an [nlobjSubList](#) object by name

Parameters

- **name** {string} [required] - The internal ID name of the sublist. Internal ID names must be in lowercase and contain no spaces.

Returns

- [nlobjSubList](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getSubTab(name)

Returns an [nlobjTab](#) object by name

Parameters

- **name** {string} [required] - The internal ID name of the subtab. Internal ID names must be in lowercase and contain no spaces.

Returns

- [nlobjTab](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getTab(name)

Returns an [nlobjTab](#) object by name

Parameters

- **name** {string} [required] - The internal ID name of the tab. Internal ID names must be in lowercase and contain no spaces.

Returns

- [nlobjTab](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getTabs()

Returns an array of [nlobjTab](#) objects containing all the tabs in a form.

Returns

- [nlobjTab](#)[]

Since

- Version 2012.2

Example

This sample shows how to create a new form using `nlapicreateForm`, add two tabs to the form using `nlobjForm.addTab`, add one field to each new tab using `nlobjForm.addField`, and then get all tabs in the new form using `nlobjForm.getTabs`.

```
//Create a form called Interns.
var newForm = nlapicreateForm('Interns');

//Add a tab to the Interns form.
var firstTab = newForm.addTab('custpage_academichistorytab', 'Academic History');

//Add a text field to the first tab.
newForm.addField('custpage_universityname', 'text', 'University Name', null, 'custpage_academichistorytab');

//Add a second tab to the Interns form.
var secondTab = newForm.addTab('custpage_studentcontacttab', 'Student Contact');

//Add an email field to the second tab.
newForm.addField('custpage_studentemail', 'email', 'Student Email', null, 'custpage_studentcontacttab');

//Get all tabs in the Interns form.
var tabs = newForm.getTabs();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

insertField(field, nextfld)

Inserts a field (`nlobjField`) in front of another field and returns a reference to it

Parameters

- **field** {`nlobjField`} [required] - `nlobjField` object to insert
- **nextfield** {string} [required] - The name of the field you are inserting in front of

Returns

- `nlobjField`

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

insertSubList(sublist, nextsub)

Inserts a sublist (`nlobjSubList`) in front of another sublist/subtab and returns a reference to it

Parameters

- **sublist** {`nlobjSubList`} [required]- `nlobjSubList` object to insert

- **nextsub** {string} [required] - The internal ID name of the sublist/subtab you are inserting in front of

Returns

- [nlobjSubList](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

insertSubTab(subtab, nextsub)

Inserts a subtab (**nlobjTab**) in front of another sublist/subtab and returns a reference to it

Parameters

- **name** {string} [required] - The internal ID name of the subtab. Internal ID names must be in lowercase and contain no spaces.
- **nextsub** {string} [required] - The name of the sublist/subtab you are inserting in front of

Returns

- [nlobjTab](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

insertTab(tab, nexttab)

Inserts a tab (**nlobjTab**) in front of another tab and returns a reference to it

Parameters

- **tab** {nlobjTab} [required] - [nlobjTab](#) object to insert
- **nexttab** {string} [required] - The tab name for the tab you are inserting in front of

Returns

- [nlobjTab](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

removeButton(name)

Removes an [nlobjButton](#) object. This method can be used on custom buttons and certain built-in NetSuite buttons. For a list of built-in buttons that support this method, see the list of buttons in the section [Button IDs](#) in the NetSuite Help Center.

Parameters

- **name** {string} [required] - The internal ID of the button to be removed. Internal IDs must be in lowercase and contain no spaces.

Returns

- void

Since

- Version 2008.2

Example

```
function removeUpdateOrderButton(type, form)
{
    form.removeButton('custpage_updateorderbutton');
}
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setFieldValues(values)

Sets the values of multiple fields on the current form. This API can be used in beforeLoad scripts to initialize field scripts on new records or non-stored fields. (See the help topic [User Event beforeLoad Operations](#) in the NetSuite Help Center for information on beforeLoad user event triggers.)

Parameters

- **values** {hashtable<string, string>} [required] - An associative array containing name/value pairs, which maps field names to field values

Returns

- void

Since

- Version 2008.2

Example

```
var form = nlapiCreateForm( 'Tax Form' );
form.addField('name', 'text', 'Name');
form.addField('email', 'email', 'Email');
```

```
form.addField('phone', 'phone', 'Phone');
form.setFieldValues({ name: 'Jane', email: 'JWolfe@netsuite.com', phone: '650.123.4567' })
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setScript(script)

Sets the Client SuiteScript file used for this form

Parameters

- **script** {string | int} [required] - The scriptId or internal ID for the global client script used to enable Client SuiteScript on this form

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setTitle(title)

Sets the title for this form

Parameters

- **title** {string} [required] - The title used for this form

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjList

Primary object used to encapsulate a list page. Note that the [nlapiCreateList\(title, hideNavbar\)](#) function returns a reference to this object.

Methods

- [addButton\(name, label, script\)](#)
- [addColumn\(name, type, label, align\)](#)

- `addEditColumn(column, showView, showHrefCol)`
- `addPageLink(type, title, url)`
- `addRow(row)`
- `addRows(rows)`
- `setScript(script)`
- `setStyle(style)`
- `setTitle(title)`

addButton(name, label, script)

Adds an `nlobjButton` object to the footer of the page

Parameters

- **name** {string} [required] - The internal ID name of the button. Internal ID names must be in lowercase and contain no spaces. For example, if you add a button that appears on the UI as **Update Order**, the internal ID should be something equivalent to *updateorder*.
- **type** {string} [required] - The UI label used for this button
- **script** {string} [optional] - The onclick button script function name

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addColumn(name, type, label, align)

Adds an `nlobjColumn` object to a list and returns a reference to this column

Parameters

- **name** {string} [required] - The internal ID name of this column. Note that internal ID names must be in lowercase and contain no spaces.
- **type** {string} [required]- The field type for this column. Use any of the following field types:
 - text
 - email
 - phone
 - date
 - currency
 - float
 - integer
 - select

- url
- timeofday
- textarea
- percent
- inlinehtml
- **label** {string} [required] - The UI label for this column
- **align** {string} [optional] - The layout justification for this column. Possible values include:
 - center
 - right
 - left (default)

Returns

- [nlobjColumn](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addEditColumn(column, showView, showHrefCol)

Adds an Edit or Edit/View column to Portlets (created with the [nlobjPortlet](#) object) and Suitelet and Portlet lists (created with the [nlobjList](#) object). Note that the Edit or Edit/View column will be added to the left of a previously existing column.

This figure shows Edit | View links added to a Portlet. These links appear to the left of the Due Date column.

NEW	EDIT VIEW	DUE DATE	TASK TITLE	COMPANY	PRIORITY	STATUS
	Edit View	9/24/2015	Create an estimate for CTA		Medium	Not Started
	Edit View	9/25/2015	Draft Product Expansion Needs		Medium	Not Started
	Edit View	9/26/2015	Review E-Auctions Online Proposal		Medium	Not Started
	Edit View	9/27/2015	Re-negotiate Insurance Rates		Medium	Not Started
	Edit View	9/28/2015	Call Lawyers re: Telco Contract		Medium	Not Started

Parameters

- **column** {nlobjColumn} [required] - An [nlobjColumn](#) object to the left of which the Edit/View column will be added
- **showView** {boolean} [optional] - If *true* then an Edit/View column will be added. Otherwise only an Edit column will be added.
- **showHrefCol** {boolean} [optional] - If set, this value must be included in row data provided for the list and will be used to determine whether the URL for this link is clickable (specify T for clickable, F for non-clickable)

Returns

- [nlobjColumn](#)

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addPageLink(type, title, url)

Adds a navigation cross-link to the list page

Parameters

- **type** {string} [required] - The type of navbar link to add. Use any of the following types:
 - **breadcrumb** - appears on top-left corner after system bread crumbs
 - **crosslink** - appears on top-right corner
- **title** {string} [required] - The UI text displayed in the link
- **url** {string} [required] - The URL used for this link

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRow(row)

Adds a row (Array of name/value pairs or **nlobjSearchResult**) to this portlet.

Parameters

- **row** {hashtable<string, string> | nlobjSearchResult} [required] - An Array of rows containing name/value pairs containing the values for corresponding [nlobjColumn](#) objects in this list -or- an [nlobjSearchResult](#). Note that several special fields: recordtype, id, and fieldname_display (UI display value for select fields) are automatically added for each search result.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRows(rows)

Adds multiple rows (Array of **nlobjSearchResult** objects or name/value pair Arrays) to a portlet.

Parameters

- **rows** {hashtable<string, string>[] | nlobjSearchResult[]} [required] - An Array of Arrays containing name/value pairs containing column values for multiple rows -or- an Array of **nlobjSearchResult** objects containing the results of a search with columns matching the columns on the list.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setScript(script)

Sets the Client SuiteScript used for this page.

Parameters

- **script** {string, int} [required] - scriptId or internal ID for global client script used to enable Client SuiteScript on page

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setStyle(style)

Sets the display style for this list

Parameters

- **style** {string} [required] - The display style value. Use any of the following styles:
 - **grid**
 - **report**
 - **plain**
 - **normal**

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setTitle(title)

Sets the title for this list

Parameters

- **title** {string} [required] - The title for a list

Returns

- void


Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjPortlet

Primary object used to encapsulate scriptable dashboard portlets. Using SuiteScript you can create a LIST, FORM, HTML, or LINKS type of portlet.

 **Note:** In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for definitions and examples of each portlet type. This section also describes how to set the portlet type on the Script page in the NetSuite UI.

To create a portlet using SuiteScript, pass the **portlet** and **column** arguments to your user-defined function. The system then automatically instantiates a nlobjPortlet object (via the **portlet** argument) and provides a placeholder for you to specify the portlet's column position on the NetSuite dashboard (via the **column** argument). Available column position values are **1** = left column, **2** = middle column, **3** = right column.

The following is an example of a user-defined function that includes the **portlet** and **column** arguments:

```
function myPortlet( portlet, column )
{
    portlet.setTitle('Portlet Title');
    portlet.addLine('This is my SuiteScript portlet', null, 1);
}
```

Note: argument is optional. If you do not plan on setting a column position value, you do not need to pass the `column` argument. For example:

```
function myPortlet( portlet )
{
    portlet.setTitle('Portlet Title');
    portlet.writeLine('This is my SuiteScript portlet', null, 1);
}
```

After you have instantiated a portlet object, use any of the following methods to set or add values.

nlobjPortlet Methods

- `addColumn(name, type, label, just)`
- `addEditColumn(column, showView, showHrefCol)`
- `addField(name, type, label, source)`
- `addLine(text, url, indent)`
- `addRow(row)`
- `addRows(rows)`
- `setHtml(html)`
- `setRefreshInterval(n)`
- `setScript(scriptid)`
- `setSubmitButton(url, label, target)`
- `setTitle(title)`

addColumn(name, type, label, just)

Adds an `nlobjColumn` object to a list and returns a reference to this column. Note that this API is only available if the portlet type is a **LIST** type. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **name** {string} [required] - The internal ID name of this column. Internal ID names must be in lowercase and contain no spaces.
- **type** {string} [required] - The field type for this column. Use any of the following field types:
 - text
 - email
 - phone
 - date
 - currency
 - float
 - integer
 - checkbox
 - select

- url
- timeofday
- textarea
- percent
- inlinehtml
- **label** {string} [required] - The UI label for this column
- **just** {string} [optional] - The layout justification for this column. Use any of the following layout types:
 - center
 - right
 - left - (default)

Returns

- [nlobjColumn](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addEditColumn(column, showView, showHrefCol)

Adds an Edit or Edit | View column to **LIST** portlets (see figure). This method can also be used with [nlobjList](#) when creating Suitelet lists and portlet lists. Note that the Edit or Edit | View column will be added to the left of a previously existing column.

The following figure shows Edit | View links added to a portlet. These links appear to the left of the Due Date column.

NEW	EDIT VIEW	DUE DATE	TASK TITLE	COMPANY	PRIORITY	STATUS
	Edit View	9/24/2015	Create an estimate for CTA		Medium	Not Started
	Edit View	9/25/2015	Draft Product Expansion Needs		Medium	Not Started
	Edit View	9/26/2015	Review E-Auctions Online Proposal		Medium	Not Started
	Edit View	9/27/2015	Re-negotiate Insurance Rates		Medium	Not Started
	Edit View	9/28/2015	Call Lawyers re: Telco Contract		Medium	Not Started

Parameters

- **column** {nlobjColumn} [required] - An [nlobjColumn](#) object to the left of which the Edit | View column will be added
- **showView** {boolean} [optional] - If **true** then an Edit | View column will be added. Otherwise only an Edit column will be added.
- **showHrefCol** {string} [optional] - If set, this value must be included in row data provided for the list and will be used to determine whether the URL for this link is clickable (specify **T** for clickable, **F** for non-clickable)

Returns

- [nlobjColumn](#)

Since

- Version 2008.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addField(name, type, label, source)

Adds an [nlobjField](#) object to a portlet and returns a reference to it.

This API is only available if the portlet type is **FORM**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **name** {string} [required] - The internal ID name of this field. Internal ID names must be in lowercase and contain no spaces.
- **type** {string} [required] - The field type for this field. Use any of the following fields types:
 - text
 - email
 - phone
 - date
 - currency
 - float
 - integer
 - checkbox
 - select
 - url
 - timeofday
 - textarea
 - percent
 - inlinehtml
- **label** {string} [required] - The UI label for this field
- **source** {int | string} [optional] - The internalId or scriptId of the source list for this field if it's a select (List/Record) field, or radio value for radio fields. In the NetSuite Help Center, see [List/Record Type IDs](#) for the internal IDs of all supported list/record types.



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with [nlobjField.addSelectOption\(value, text, selected\)](#). The select values are determined by the source record or list.

Returns

- [nlobjField](#)

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addLine(text, url, indent)

Adds a line (containing text or basic HTML) with optional indenting and URL to a **LINKS** portlet.

This API is only available if the portlet type is **LINKS**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **text** {string} [required] - Content written to this line (can contain basic HTML formatting)
- **url** {string} [optional] - URL if this line should be clickable (if NULL then line will not be clickable)
- **indent** {int} [optional] - Indent level used for this line. Valid values are 0 to 5.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRow(row)

Adds a row (**nlobjSearchResult** or Array of name/value pairs) to a LIST portlet.

This API is only available if the portlet type is **LIST**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **row** {hashtable<string, string> | nlobjSearchResult} [required] - An Array of rows containing name/value pairs containing the values for corresponding [nlobjColumn](#) objects in this list -or- an [nlobjSearchResult](#). Note that several special fields: recordtype, id, and fieldname_display (display value for select fields) are automatically added for each search result.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRows(rows)

Adds multiple rows (Array of **nlobjSearchResult** objects or name/value pair Arrays) to a **LIST** portlet.

This API is only available if the portlet type is **LIST**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **rows** {hashtable<string, string>[] | nlobjSearchResult[]} [required] - An Array of Arrays containing name/value pairs containing column values for multiple rows -or- an Array of [nlobjSearchResult](#) objects containing the results of a search with columns matching the columns on the list.

Returns

- void



Important: Ensure there is a search column or name/value pair that corresponds to every column added to this portlet.

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHtml(html)

Sets the entire content of an HTML portlet (content will be placed inside <TD>...</TD> tags).

This API is only available if the portlet type is **HTML**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **html** {string} [required] - Raw HTML containing the contents of an HTML portlet. The content must start and end with a TD tag.



Note: The recommended approach is to wrap the interior content inside an HTML container such as a DIV, TABLE, or SPAN.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setRefreshInterval(n)

Sets the regular interval when a FORM portlet automatically refreshes itself.

This API is only available if the portlet type is **FORM**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **n** {int} [required] - Number of seconds. In production mode, this value must be at least 60 seconds. An error is raised if this value is less than zero, and in production if it is less than 60.

Returns

- void

Since

- Version 2011.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setScript(scriptId)

Sets the client-side script for a FORM portlet. For example, you can use this method to call a script to implement client-side validation, dynamically calculate field totals, and change data based on the value of another field. Note that you can only set one script. Setting another script implicitly removes the previous script.

This API is only available if the portlet type is **FORM**. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **scriptId** {int | string} [required] - The script internalId or custom scriptId of a record-level client script. Scripts of this type are deployed globally and run against an entire record type. For more information, see the help topic [Form-level and Record-level Client Scripts](#).

Returns

- void

Since

- Version 2011.1

Example

For an example use of this method, see the example for [nlapiResizePortlet\(\)](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setSubmitButton(url, label, target)

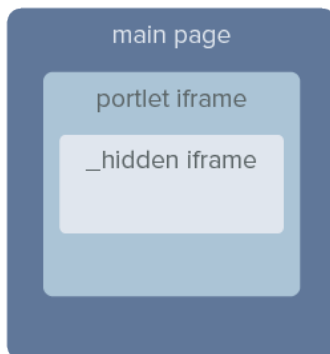
Adds a SUBMIT button with an optional custom label to this FORM portlet.

This API is only available if the portlet type is a **FORM** type. (In the NetSuite Help Center, see the help topic [Portlet Scripts](#) for portlet type definitions. This section also shows how to define your portlet type on the portlet Script record page in the NetSuite UI.)

Parameters

- **url** {string} [required] - The URL that the FORM will be POST-ed to when the user clicks this submit button
- **label** {string} [optional] - The UI label used for displaying this button. If a value is not specified, the default value is **Save**.
- **target** {string} [optional] - The target attribute of the portlet's FORM element, if it is different from the portlet's own embedded iframe. Supported values include standard HTML target attributes such as `'_top'`, `'_parent'`, and `'_blank'`, frame names, and the special NetSuite-specific identifier `'_hidden'`.

Setting the target to `'_hidden'` allows submission to a backend that returns results to a hidden child iframe within the portlet's embedded iframe, so that these results do not replace portlet content. For example, a custom form portlet could submit to a backend suitelet, and if the suitelet returns an error, it is displayed in the hidden child iframe and does not change other portlet contents.



The following code provides an example:

```
portlet.setSubmitButton(nlapiResolveURL('SUITELET', 'customscript_suitelet', 'customdeploy_suitelet'),
    'Save', '_hidden');
```

Note: The **target** parameter was added as of Version 2011.1.

Returns

- `nlobjButton`

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setTitle(title)

Sets the portlet title

Parameters

- **title** {string} [required] - The title used for this portlet

Returns

- void

Since

- Version 2008.2

Example

```
function demoSimpleFormPortlet(portlet, column)
{
    portlet.setTitle('Simple Form Portlet')

    //remainder of code...
}
```

The screenshot displays a NetSuite user interface. At the top is a navigation bar with tabs: Activities, Payments, Transactions, Lists, Reports, Customization, Documents, Setup, Training, and Support. Below this is a 'Home' section with a 'Flash Portlet' containing a logo for 'Christy's Catering'. To the right is an 'Estimates List Detail' table. In the bottom left, a 'Simple Form Portlet' is highlighted with a red border. This portlet contains the following form elements: a TEXT input field, an INTEGER input field, a DATE input field, a SELECT dropdown menu currently showing 'Oranges', and a TEXTAREA. A blue 'Submit' button is at the bottom of the form. To the right of the form portlet is a 'Calendar: My Calendar' showing the month of August 2015.

NUMBER	DATE	CUSTOMER	SALES REP	AMOUNT
EST10001	2/8/2003	Abe Simpson	Jon Baker	14,017.00
EST10002	4/1/2003	Elijah Tuck	Poncho Poncherelli	2,718.19
EST10003	4/16/2003	Kent Brockman	Jon Baker	3,454.00
EST10004	5/10/2003	Larry Smith	Jon Baker	5,926.45
EST10005	8/1/2003	Lionel Hutz	Jon Baker	2,028.65
EST10006	9/26/2003	Milhouse Van Houten	Jon Baker	3,150.00
EST10017	1/5/2004	Barry Springsteen	Jon Baker	3,727.58
EST10008	1/6/2004	Chip Matthews	Mathew Christner	3,600.00
EST10018	1/6/2004	Adina Fitzpatrick	Luke Duke	1,125.00
EST10019	1/6/2004	Andrew Kuykendall	Poncho Poncherelli	16,723.38
EST10009	1/8/2004	Aaron Rosewall-Godley	Mathew Christner	5,829.44
EST10014	1/8/2004	Cesar Petras	Theodore Hosch	7,288.39
EST10012	1/9/2004	Damon Hovanec	Mathew Christner	6,650.00
EST10007	1/10/2004	Cameron Sardella	Jon Baker	480,500.00
EST10013	1/12/2004	Cesar Petras	Theodore Hosch	518,880.00

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjSubList

Primary object used to encapsulate a NetSuite sublist. This object is read-only except for instances created via the UI Object API using Suitelets or beforeLoad user event scripts.

To add a sublist, you must first create a custom form using `nlapicreateForm(title, hideNavbar)`, which returns an `nlobjForm` object.

After the form object is instantiated, you can add a new sublist to the form using the `nlobjForm.addSubList(name, type, label, tab)` method, which returns a reference to `nlobSublist`.



Important: When you edit a sublist line with SuiteScript, it triggers an internal validation of the sublist line. If the line validation fails, the script also fails. For example, if your script edits a closed catch up period, the validation fails and prevents SuiteScript from editing the closed catch up period.

nlobjSubList Methods

- `addButton(name, label, script)`
- `addField(name, type, label, source)`
- `addMarkAllButtons()`
- `addRefreshButton()`
- `getLineItemCount()`
- `getLineItemValue(group, fldnam, linenum)`
- `setAmountField(field)`
- `setDisplayType(type)`
- `setHelpText(help)`
- `setLabel(label)`
- `setLineItemValue(name, linenum, value)`
- `setLineItemValues(values)`
- `setUniqueField(name)`

addButton(name, label, script)

Adds a button to a sublist

Parameters

- **name** {string} [required] - The internal ID name of the button. Internal ID names must be in lowercase and contain no spaces.
- **type** {string} [required] - The UI label for the button
- **script** {string} [optional] - The onclick script function name

Returns

- `nlobjButton`

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addField(name, type, label, source)

Adds a field (column) to a sublist

Parameters

- **name** {string} [required] - The internal ID name of the field. Internal ID names must be in lowercase and contain no spaces.
- **type** {string} [required] - The field type for this field. Use any of the following types:
 - text
 - email
 - phone
 - date
 - datetimetz - This field type lets you combine date and time values in one field. For example, you may want a single field to contain date and time “timestamp” data. After a user enters a date/time value, the data is rendered in the user's preferred date and time format, as well as the user's time zone. Also note that time values are stored in NetSuite down to the second.
 - currency
 - float
 - integer
 - checkbox
 - select
 - url
 - image - This field type is available **only** for fields appearing on list/staticlist sublists. You cannot specify an **image** field on a form.
 - timeofday
 - textarea
 - percent
 - radio - only supported for sublists of type *list*
- **label** {string} [required] - The UI label for this field
- **source** - {int | string} [optional] - The internalId or scriptId of the source list for this field if it's a select (List/Record) field. In the NetSuite Help Center, see [List/Record Type IDs](#) for the internal IDs of all supported list/record types.



Important: After you create a select or multi-select field that is sourced from a record or list, you cannot add additional values with `nlobjField.addSelectOption(value, text, selected)`. The select values are determined by the source record or list.

Returns

- `nlobjField`

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addMarkAllButtons()

Adds a "Mark All" and an "Unmark All" button to a sublist. Only valid on scriptable sublists of type **LIST**. Requires a check box column to exist on the form, which will be automatically checked/unchecked depending on what the end user does.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRefreshButton()

Adds a Refresh button to sublists of type *list* or **staticlist** to auto-refresh the sublist if its contents are dynamic. In this case, the sublist is refreshed without having to reload the contents of the entire page.

Returns

- nlobjButton


Since

- Version 2009.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemCount()

Returns the number of lines on a sublist

 **Important:** The first line number on a sublist is **1** (not 0).

Returns

- The integer value of the number of line items on a sublist

Example

```
function request(request, response)
{
  var form = nlapiCreateForm('myform');
  var list = form.addSubList('results', 'staticlist', 'My Sublist', 'resultsTab');
  list.addField("rownum", "text", "#");

  var i=1;
  for(var i=1; i< 10; i++ )
    list.setLineItemValue("rownum", i, "val"+i);
}
```

```

var count = list.getLineItemCount();
form.addSubmitButton("Count#:"+count);


response.writePage( form );
}

```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

getLineItemValue(group, fldnam, linenum)

Returns string value of a sublist field. Note that you cannot set default line item values when the line is not in edit mode.

 **Note:** Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Parameters

- **group** {string} [required] - The sublist internal ID (for example, use **addressbook** as the ID for the Address sublist). See Using the SuiteScript Records Browser for sublists that support SuiteScript, sublist internal IDs, and sublist field IDs.
- **fldnam** {string} [required] - The internal ID of the field (line item) whose value is being returned
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).

Returns

- The string value of a sublist line item field

Since

- Version 2010.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setAmountField(field)

Designates a particular column as the totalling column, which is used to calculate and display a running total for the sublist

Parameters

- **field** {string} [required] - The internal ID name of the field on this sublist used to calculate running total

Returns

- void

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setDisplayType(type)

Sets the display style for this sublist. This method is only supported on scripted or staticlist sublists via the UI Object API.

Parameters

- **type** {string} [required] - The display type for this sublist. Use either of the following two values:
 - hidden
 - normal - (default)

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHelpText(help)

Adds inline help text to this sublist. This method is only supported on sublists via the UI Object API.

Parameters

- **help** {string} [required] - Inline help text used for this sublist

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLabel(label)

Sets the label for this sublist. This method is only supported on sublists via the UI Object API.

Parameters

- **label** {string} [required] - The UI label for this sublist

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLineItemValue(name, linenum,value)

Sets the value of a cell in a sublist field.

Parameters

- **name** {string} [required] - The internal ID name of the line item field being set
- **linenum** {int} [required] - The line number for this field. Note the first line number on a sublist is **1** (not 0).
- **value** {string} [required] - The value the field is being set to

Returns

- void



Tip: Normally custom transaction column fields that are not checked to show on a custom form are not available to get/setLineItemValue APIs. However, if you set them to show, but then set the label to empty, they will be available on the form but will not appear on the sublist. Note this does not apply to fields that are marked as Hidden on the custom field definition. These fields are always available on every form.

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setLineItemValues(values)

Sets values for multiple lines (Array of **nlobjSearchResult** objects or name/value pair Arrays) in a sublist.

Parameters

- **values** {nlobjSearchResult[] | hashtable<string, string>[]} [required] - An Array of Arrays containing name/value pairs containing column values for multiple rows -or- an Array of **nlobjSearchResult** objects containing the results of a search with columns matching the fields on the sublist. Note that several special fields: recordtype, id, and fieldname_display (UI display value for select fields) are automatically added for each search result.

Returns

- void

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setUniqueField(name)

Use this method to designate that a certain field on a sublist must contain a unique value. This method is available on inlineeditor and editor sublists only.

Parameters

- **name** {string} [required] - The internal ID of the sublist field that you want to make unique

Returns

- nlobjField

Since

- Version 2009.2

Example

The following sample shows an instance of a new Contacts sublist. Four line item fields are added to the sublist. Use `setUniqueField(name)` to set the **name** field as unique. This means, for example, that a user will not be able to enter two contacts that have a name of "Joe Smith," since the value of the **name** field must be unique.

```
var sublist = assistant.addSubList("contacts", "inlineeditor", "Contacts")
sublist.addField("name", "text", "Name");
sublist.addField("phone", "phone", "Phone");
sublist.addField("email", "email", "E-mail");
sublist.addField("address", "textarea", "Address");
sublist.setUniqueField("name");
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjTab

Primary object used to encapsulate tabs and subtabs. Note that to add a tab or subtab, you must first create a custom form using `nlapiCreateForm(title, hideNavbar)`, which returns an `nlobjForm` object.

After the form object is instantiated, you can add a new tab or subtab to the form using the `nlobjForm.addTab(name, label)` or `nlobjForm.addSubTab(name, label, tab)` methods, which both return a reference to `nlobjTab`.

Use the following `nlobjTab` methods to set tab values.

Methods

- [setLabel\(label\)](#)
- [setHelpText\(help\)](#)

setLabel(label)

Sets the tab UI label

Parameters

- **label** {string} [required] - The UI label used for this tab or subtab

Returns

- nlobjTab

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

setHelpText(help)

Sets the inline help used for this tab or subtab

Parameters

- **help** {string} [required] - Inline help used for this tab or subtab

Returns

- nlobjTab

Since

- Version 2008.2

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

nlobjTemplateRenderer

Template engine that produces HTML and PDF printed forms utilizing advanced PDF/HTML template capabilities. This object uses FreeMarker syntax to interpret a template passed in as string. Interpreted content can be rendered in two different formats: as HTML output to an [nlobjResponse](#) object or as XML string that can be passed to [nlapiXMLToPDF\(xmlstring\)](#) to produce a PDF.

This object is available when the Advanced PDF/HTML Templates feature is enabled. For information about this feature, see the help topic [Advanced PDF/HTML Templates](#).



Note: The advanced template API expects your template string to conform to FreeMarker syntax. FreeMarker documentation is available from <http://freemarker.sourceforge.net/docs/index.html>.



Note: As stated above, the [nlapiXMLToPDF](#) API can be used to produce a PDF from the string rendered by this object's methods. This API is used in conjunction with the Big Faceless Report Generator, a third-party library built by Big Faceless Organization (BFO). See [nlapiXMLToPDF\(xmlstring\)](#) for links to BFO documentation.

Use the following **nlobjTemplateRenderer** methods to produce printed forms.

Methods

- [setTemplate\(template\)](#)
- [addRecord\(var, record\)](#)
- [addSearchResults\(var, searchResult\)](#)
- [renderToString\(\)](#)
- [renderToResponse\(\)](#)

Examples

The following sample transforms a sales order into HTML which is written to an HTTP response object:

```
function renderRecord(request, response)
{
  var salesOrderID = 3;
  var salesOrder = nlapiLoadRecord('salesorder', salesOrderID);
  var renderer = nlapiCreateTemplateRenderer();
  renderer.setTemplate('.');
  renderer.addRecord('record', salesOrder);
  response.setContentType('HTMLDOC');
  renderer.renderToResponse(response);
}
```

The following sample transforms search results into HTML which is written to an HTTP response object:

```
function renderSearchResults(request, response)
{
  var searchID = 3;
  var runTimeFilter = new nlobjSearchFilter(...);
  var results = nlapiSearchRecord(null, searchID, runTimeFilter, null);
  var renderer = nlapiCreateTemplateRenderer();
  renderer.setTemplate('.');
  renderer.addSearchResults('results', results);
  response.setContentType('HTMLDOC');
  renderer.renderToResponse(response);
}
```

The following sample transforms a sales order into XML, which is further transformed by [nlapiXMLToPDF](#) to produce an [nlobjFile](#) with PDF file type:

```
function renderInlinePDF(request, response)
{
  var salesOrderID = 3;
  var salesOrder = nlapiLoadRecord('salesorder', salesOrderID);
  var renderer = nlapiCreateTemplateRenderer();
  renderer.setTemplate('.');
  renderer.addRecord('record', salesOrder);
  var xml = renderer.renderToString();
  var file = nlapiXMLToPDF(xml);
  response.setContentType('PDF', 'SORD'+salesOrder.getFieldValue('tranid')+'.pdf', 'inline');
  response.write(file.getValue());
}
```

Note that the (...) parameter for **setTemplate** represents the raw template string.

You can also use `nlobjTemplateRenderer` to print a large volume of documents. For more information, see the help topic [Using SuiteScript for Transaction Records](#).

Note: See the help topic [Using SuiteScript to Apply Advanced Templates to Non-Transaction Records](#) for a code sample and an explanation of how to use this object to print a record type that is not a transaction.

setTemplate(template)

Passes in raw string of template to be transformed by FreeMarker.

Parameters

- **template** {string} [required] - raw string of template

Returns

- void

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addRecord(var, record)

Binds `nlobjRecord` object to variable name used in template.

Parameters

- **var** {string} [required] - variable name that represents record
- **record** {nlobjRecord} [required] - NetSuite record

Returns

- void

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

addSearchResults(var, searchResult)

Binds `nlobjSearchResult` object to variable name used in template.

Parameters

- **var** {string} [required] - variable name that represents search result

- **searchResult** {nlobjSearchResult} [required] - NetSuite search result

Returns

- void

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

renderToString()

Returns template content interpreted by FreeMarker as XML string that can be passed to [nlapiXMLToPDF\(xmlstring\)](#) to produce PDF output.



Note: The nlapiXMLToPDF API is used in conjunction with the Big Faceless Report Generator, a third-party library built by Big Faceless Organization (BFO). See [nlapiXMLToPDF\(xmlstring\)](#) for links to BFO documentation.

Parameters

- none

Returns

- XML string of template interpreted by FreeMarker

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

renderToResponse()

Renders HTML output to [nlobjResponse](#) object.

Parameters

- none

Returns

- HTML output to an HTTP response object

Since

- Version 2013.1

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

SuiteScript 1.0 API - Alphabetized Index

The following is an alphabetized list of all SuiteScript functions and objects. Click these links to see either [Functions](#) or [Objects](#).

For a task-based grouping of all APIs, see [SuiteScript Functions](#). For a high-level overview of the SuiteScript API, see [SuiteScript 1.0 API Overview](#).

Functions

- [nlapiAddDays\(d, days\)](#)
- [nlapiAddMonths\(d, months\)](#)
- [nlapiAttachRecord\(type, id, type2, id2, attributes\)](#)
- [nlapiCalculateTax\(\)](#)
- [nlapiCancelLineItem\(type\)](#)
- [nlapiCommitLineItem\(type\)](#)
- [nlapiCopyRecord\(type, id, initializeValues\)](#)
- [nlapiCreateAssistant\(title, hideHeader\)](#)
- [nlapiCreateCSVImport\(\)](#)
- [nlapiCreateCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiCreateSearch\(type, filters, columns\)](#)
- [nlapiCreateSubrecord\(fldname\)](#)
- [nlapiCreateError\(code, details, suppressNotification\)](#)
- [nlapiCreateFile\(name, type, contents\)](#)
- [nlapiCreateForm\(title, hideNavbar\)](#)
- [nlapiCreateList\(title, hideNavbar\)](#)
- [nlapiCreateRecord\(type, initializeValues\)](#)
- [nlapiCreateReportDefinition\(\)](#)
- [nlapiCreateReportForm\(title\)](#)
- [nlapiCreateTemplateRenderer\(\)](#)
- [nlapiDateToString\(d, format\)](#)
- [nlapiDeleteFile\(id\)](#)
- [nlapiDeleteRecord\(type, id, initializeValues\)](#)
- [nlapiDetachRecord\(type, id, type2, id2, attributes\)](#)
- [nlapiDisableField\(fldnam, val\)](#)
- [nlapiDisableLineItemField\(type, fldnam, val\)](#)
- [nlapiEditCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiEditSubrecord\(fldname\)](#)
- [nlapiEncrypt\(s, algorithm, key\)](#)
- [nlapiEscapeXML\(text\)](#)
- [nlapiExchangeRate\(sourceCurrency, targetCurrency, effectiveDate\)](#)
- [nlapiFindLineItemMatrixValue\(type, fldnam, val, column\)](#)

- `nlapiFindLineItemValue(type, fldnam, val)`
- `nlapiFormatCurrency(str)`
- `nlapiGetContext()`
- `nlapiGetCurrentLineItemDateTimeValue(type, fieldId, timeZone)`
- `nlapiGetCurrentLineItemIndex(type)`
- `nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)`
- `nlapiGetCurrentLineItemText(type, fldnam)`
- `nlapiGetCurrentLineItemValue(type, fldnam)`
- `nlapiGetCurrentLineItemValues(type, fldnam)`
- `nlapiGetDateTimeValue(fieldId, timeZone)`
- `nlapiGetDepartment()`
- `nlapiGetJobManager(jobType)`
- `nlapiGetField(fldnam)`
- `nlapiGetFieldText(fldnam)`
- `nlapiGetFieldTexts(fldnam)`
- `nlapiGetFieldValue(fldnam)`
- `nlapiGetFieldValues(fldnam)`
- `nlapiGetLineItemCount(type)`
- `nlapiGetLineItemDateTimeValue(type, fieldId, lineNum, timeZone)`
- `nlapiGetLineItemField(type, fldnam, linenum)`
- `nlapiGetLineItemMatrixField(type, fldnam, linenum, column)`
- `nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)`
- `nlapiGetLineItemText(type, fldnam, linenum)`
- `nlapiGetLineItemValue(type, fldnam, linenum)`
- `nlapiGetLineItemValues(type, fldname, linenum)`
- `nlapiGetLocation()`
- `nlapiGetLogin()`
- `nlapiGetMatrixCount(type, fldnam)`
- `nlapiGetMatrixField(type, fldnam, column)`
- `nlapiGetMatrixValue(type, fldnam, column)`
- `nlapiGetNewRecord()`
- `nlapiGetOldRecord()`
- `nlapiGetRecordId()`
- `nlapiGetRecordType()`
- `nlapiGetRole()`
- `nlapiGetSubsidiary()`
- `nlapiGetUser()`
- `nlapiInitiateWorkflow(recordtype, id, workflowid, initialvalues)`
- `nlapiInsertLineItem(type, line)`
- `nlapiInsertLineItemOption(type, fldnam, value, text, selected)`
- `nlapiInsertSelectOption(fldnam, value, text, selected)`

- [nlapiIsLineItemChanged\(type\)](#)
- [nlapiLoadFile\(id\)](#)
- [nlapiLoadRecord\(type, id, initializeValues\)](#)
- [nlapiLoadSearch\(type, id\)](#)
- [nlapiLogExecution\(type, title, details\)](#)
- [nlapiLookupField\(type, id, fields, text\)](#)
- [nlapiMergeRecord\(id, baseType, baseId, altType, altId, fields\)](#)
- [nlapiMergeTemplate\(id, baseType, baseId, altType, altId, fields\)](#)
- [nlapiOutboundSSO\(id\)](#)
- [nlapiPrintRecord\(type, id, mode, properties\)](#)
- [nlapiRefreshLineItems\(type\)](#)
- [nlapiRefreshPortlet\(\)](#)
- [nlapiRemoveCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiRemoveLineItem\(type, line\)](#)
- [nlapiRemoveLineItemOption\(type, fldnam, value\)](#)
- [nlapiRemoveSelectOption\(fldnam, value\)](#)
- [nlapiRemoveSubrecord\(fldname\)](#)
- [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#)
- [nlapiRequestURLWithCredentials\(credentials, url, postdata, headers, httpsMethod\)](#)
- [nlapiResizePortlet\(\)](#)
- [nlapiResolveURL\(type, identifier, id, displayMode\)](#)
- [nlapiScheduleScript\(scriptId, deployId, params\)](#)
- [nlapiSearchDuplicate\(type, fields, id\)](#)
- [nlapiSearchGlobal\(keywords\)](#)
- [nlapiSearchRecord\(type, id, filters, columns\)](#)
- [nlapiSelectLineItem\(type, linenum\)](#)
- [nlapiSelectNewLineItem\(type\)](#)
- [nlapiSelectNode\(node, xpath\)](#)
- [nlapiSelectNodes\(node, xpath\)](#)
- [nlapiSelectValue\(node, xpath\)](#)
- [nlapiSelectValues\(node, path\)](#)
- [nlapiSendCampaignEmail\(campaigneventid, recipientid\)](#)
- [nlapiSendEmail\(author, recipient, subject, body, cc, bcc, records, attachments, notifySenderOnBounce, internalOnly, replyTo\)](#)
- [nlapiSendFax\(author, recipient, subject, body, records, attachments\)](#)
- [nlapiSetCurrentLineItemDateTimeValue\(type, fieldId, dateTime, timeZone\)](#)
- [nlapiSetCurrentLineItemMatrixValue\(type, fldnam, column, value, firefieldchanged, synchronous\)](#)
- [nlapiSetCurrentLineItemText\(type, fldnam, text, firefieldchanged, synchronous\)](#)
- [nlapiSetCurrentLineItemValue\(type, fldnam, value, firefieldchanged, synchronous\)](#)
- [nlapiSetCurrentLineItemValues\(type, fldnam, values, firefieldchanged, synchronous\)](#)
- [nlapiSetDateTimeValue\(fieldId, dateTime, timeZone\)](#)

- [nlapiSetFieldText\(fldname, txt, firefieldchanged, synchronous\)](#)
- [nlapiSetFieldTexts\(fldname, txts, firefieldchanged, synchronous\)](#)
- [nlapiSetFieldValue\(fldnam, value, firefieldchanged, synchronous\)](#)
- [nlapiSetFieldValues\(fldnam, value, firefieldchanged, synchronous\)](#)
- [nlapiSetLineItemDateTimeValue\(type, fieldId, lineNum, dateTime, timeZone\)](#)
- [nlapiSetLineItemValue\(type, fldnam, linenum, value\)](#)
- [nlapiSetMatrixValue\(type, fldnam, column, value, firefieldchanged, synchronous\)](#)
- [nlapiSetRecoveryPoint\(\)](#)
- [nlapiSetRedirectURL\(type, identifier, id, editmode, parameters\)](#)
- [nlapiStringToDate\(str, format\)](#)
- [nlapiStringToXML\(text\)](#)
- [nlapiSubmitCSVImport\(nlobjCSVImport\)](#)
- [nlapiSubmitField\(type, id, fields, values, doSourcing\)](#)
- [nlapiSubmitFile\(file\)](#)
- [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#)
- [nlapiTransformRecord\(type, id, transformType, transformValues\)](#)
- [nlapiTriggerWorkflow\(recordtype, id, workflowid, actionid, stateid\)](#)
- [nlapiViewCurrentLineItemSubrecord\(sublist, fldname\)](#)
- [nlapiViewLineItemSubrecord\(sublist, fldname, linenum\)](#)
- [nlapiViewSubrecord\(fldname\)](#)
- [nlapiXMLToPDF\(xmlstring\)](#)
- [nlapiXMLToString\(xml\)](#)
- [nlapiYieldScript\(\)](#)

Objects

- [nlobjAssistant](#)
- [nlobjAssistantStep](#)
- [nlobjButton](#)
- [nlobjColumn](#)
- [nlobjConfiguration](#)
- [nlobjContext](#)
- [nlobjCredentialBuilder\(string, domainString\)](#)
- [nlobjCSVImport](#)
- [nlobjError](#)
- [nlobjField](#)
- [nlobjFieldGroup](#)
- [nlobjFile](#)
- [nlobjForm](#)
- [nlobjList](#)
- [nlobjLogin](#)

- [nlobjPivotColumn](#)
- [nlobjPivotRow](#)
- [nlobjPivotTable](#)
- [nlobjPivotTableHandle](#)
- [nlobjPortlet](#)
- [nlobjRecord](#)
- [nlobjReportColumn](#)
- [nlobjReportColumnHierarchy](#)
- [nlobjReportDefinition](#)
- [nlobjReportForm](#)
- [nlobjReportRowHierarchy](#)
- [nlobjResponse](#)
- [nlobjRequest](#)
- [nlobjSearch](#)
- [nlobjSearchColumn\(name, join, summary\)](#)
- [nlobjSearchFilter](#)
- [nlobjSearchResult](#)
- [nlobjSelectOption](#)
- [nlobjSubList](#)
- [nlobjSubrecord](#)
- [nlobjTab](#)
- [nlobjTemplateRenderer](#)