

# SuiteScript Developer & Reference Guide

# General Notices

## Sample Code

NetSuite Inc. 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](http://www.netsuite.com/tos).

NetSuite 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, customers may not use the Service in excess of limits or thresholds that NetSuite considers commercially reasonable for the Service. If NetSuite reasonably concludes that a customer's use is excessive and/or will cause immediate or ongoing performance issues for one or more of NetSuite's other customers, NetSuite may slow down or throttle such customer's excess use until such time that the customer's use stays within reasonable limits. If a 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.

## Integration with Third Party Applications

NetSuite may make available to Customer certain features designed to interoperate with third party applications. To use such features, Customer may be required to obtain access to such third party applications from their providers, and may be required to grant NetSuite access to Customer's account(s) on such third party applications. NetSuite cannot guarantee the continued availability of such Service features or integration, and may cease providing them without entitling Customer to any refund, credit, or other compensation, if for example and without limitation, the provider of a third party application ceases to make such third party application generally available or available for interoperation with the corresponding Service features or integration in a manner acceptable to NetSuite.

## Copyright

This document is the property of NetSuite Inc., and may not be reproduced in whole or in part without prior written approval of NetSuite Inc. For NetSuite trademark and service mark information, see [www.netsuite.com/portal/company/trademark.shtml](http://www.netsuite.com/portal/company/trademark.shtml).

© 2016 NetSuite Inc.

# Table of Contents

Getting Started with SuiteScript .....	13
SuiteScript - The Basics .....	13
What is SuiteScript? .....	15
What can I do with the SuiteScript API? .....	15
Using the SuiteScript API with NetSuite Records .....	16
Setting Up Your SuiteScript Environment .....	19
Environment Setup Overview .....	19
Configuring NetSuite for SuiteScript .....	19
Enabling SuiteScript .....	19
Showing Record and Field IDs in Your Account .....	20
Setting Roles and Permissions for SuiteScript .....	23
Working with IDEs Other Than SuiteCloud IDE .....	24
Running a Script in NetSuite .....	27
Running Scripts in NetSuite Overview .....	27
Step 1: Create Your Script .....	28
Step 2: Add Script to NetSuite File Cabinet .....	29
Step 3: Attach Script to Form .....	30
Step 4: Create Script Record .....	32
Steps for Creating a Script Record .....	33
Creating a Custom Script Record ID .....	36
Step 5: Define Script Deployment .....	39
Steps for Defining a Script Deployment .....	39
Creating a Custom Script Deployment ID .....	40
Viewing Script Deployments .....	42
Viewing Script and Deployment System Notes .....	43
Using the Scripted Records Page .....	44
Creating Script Execution Logs .....	46
Viewing a List of Script Execution Logs .....	46
Using the Script Execution Log Tab .....	47
Using the Script Queue Monitor .....	49
Installing and Accessing the Script Queue Monitor .....	49
Script Queue Monitor User Interface .....	50
Scripting Records, Fields, Forms, and Sublists .....	55
Working with Subrecords in SuiteScript .....	55
What is a Subrecord? .....	55
Using the SuiteScript API with Subrecords .....	56
Creating and Accessing Subrecords from a Body Field .....	57
Creating and Accessing Subrecords from a Sublist Field .....	58
Setting Values on Subrecord Sublists .....	59
Saving Subrecords Using SuiteScript .....	61
Guidelines for Working with Subrecords in SuiteScript .....	61
Working with Specific Subrecords in SuiteScript .....	62
Using SuiteScript with Advanced Bin / Numbered Inventory Management .....	62
Using SuiteScript with Address Subrecords .....	71
Working with Fields .....	79
Working with Fields Overview .....	79
Referencing Fields in SuiteScript .....	80
Working with Custom Fields in SuiteScript .....	81
Working with Subtabs and Sublists .....	84
Subtabs and Sublists Overview .....	84
Subtabs and Sublists - What's the Difference? .....	84
What is a Subtab? .....	85

What is a Sublist? .....	86
Sublist Types .....	87
Editor Sublists .....	88
Inline Editor Sublists .....	88
List Sublists .....	89
Static List Sublists .....	90
Adding Subtabs with SuiteScript .....	91
Adding Sublists with SuiteScript .....	93
Working with Sublist Line Items .....	96
Adding and Removing Line Items .....	97
Getting and Setting Line Item Values .....	100
Working with Item Groups in a Sublist .....	101
Working with Sublists in Dynamic Mode and Client SuiteScript .....	101
Sublist Errors .....	103
Working with Sublists in Standard Mode and Client SuiteScript .....	103
Working with Online Forms .....	105
Inline Editing and SuiteScript .....	107
Inline Editing and SuiteScript Overview .....	107
Why Inline Edit in SuiteScript? .....	108
Inline Editing Using nlapiSubmitField .....	108
Consequences of Using nlapiSubmitField on Non Inline Editable Fields .....	109
Inline Editing (xedit) as a User Event Type .....	110
What's the Difference Between xedit and edit User Event Types? .....	110
Inline Editing and nlapiGetNewRecord() .....	111
Inline Editing and nlapiGetOldRecord() .....	111
Understanding NetSuite Script Types .....	113
Script Types Overview .....	113
SuiteScript Execution Diagram .....	115
Client Scripts .....	116
What is Client SuiteScript? .....	116
Client Script Execution .....	117
Client Event Types .....	117
Form-level and Record-level Client Scripts .....	120
Client Script Metering .....	121
Role Restrictions in Client SuiteScript .....	121
How Many Client Events Can I Execute on One Form? .....	122
Error Handling and Debugging Client SuiteScript .....	123
Client Remote Object Scripts .....	123
Running a Client Script in NetSuite .....	124
Client SuiteScript Samples .....	124
Writing Your First Client Script .....	124
Page Init Sample .....	128
Save Record Sample .....	129
Post Sourcing Sample .....	130
Validate Field Sample .....	130
Field Changed Sample .....	132
User Event Scripts .....	133
What Are User Event Scripts? .....	133
User Event Script Execution .....	134
Setting the User Event type Argument .....	135
User Event Script Execution Types .....	138
How Many User Events Can I Have on One Record? .....	140
Running a User Event Script in NetSuite .....	141
User Event Script Samples .....	141

Generating a Record Log .....	141
Creating Follow-up Phone Call Records for New Customers .....	142
Enhancing NetSuite Forms with User Event Scripts .....	144
<b>Suitelets .....</b>	<b>146</b>
What Are Suitelets? .....	146
Suitelet Script Execution .....	148
Building Custom Workflows with Suitelets .....	149
Building Suitelets with UI Objects .....	149
Backend Suitelets .....	150
Reserved Parameter Names in Suitelet URLs .....	152
SuiteScript and Externally Available Suitelets .....	152
Running a Suitelet in NetSuite .....	153
Suitelets Samples .....	154
Writing Your First Suitelet .....	154
Return a Simple XML Document .....	155
Create a Simple Form .....	156
Create a Simple List .....	157
Add a Suitelet to a Tab .....	158
Create a Suitelet Email Form .....	160
Create a Form with a URL Field .....	161
Using Embedded Inline HTML in a Form .....	162
<b>RESTlets .....</b>	<b>164</b>
Working with RESTlets .....	164
RESTlet Script Execution .....	165
Authentication for RESTlets .....	165
RESTlet URL and Domain .....	168
Using the REST roles Service to Get User Accounts, Roles, and Domains .....	169
Supported Input and Output Content Types for RESTlets .....	171
Supported Functions for RESTlets .....	171
RESTlet Governance and Session Management .....	172
RESTlet Debugging .....	172
RESTlet Error Handling .....	172
RESTlet Security .....	173
RESTlets vs. Other NetSuite Integration Options .....	173
Creating a RESTlet .....	175
Debugging a RESTlet .....	177
RESTlet HTTP Testing Tools .....	178
Sample RESTlet Code .....	178
Simple Example to Get Started .....	179
Example Code Snippets of HTTP Methods .....	179
Example RESTlet Called from a Portlet Script .....	180
Example RESTlet Request from Android .....	184
Example RESTlet Request Using nlapiRequestURL .....	184
Sample RESTlet Input Formats .....	186
RESTlet Status Codes and Error Message Formats .....	200
Tracking and Managing RESTlet Activity .....	202
<b>Scheduled Scripts .....</b>	<b>205</b>
Overview of Scheduled Script Topics .....	205
What Are Scheduled Scripts? .....	206
When Will My Scheduled Script Execute? .....	206
Deploying a Script to the Scheduling Queue .....	207
Initiating an Ad-hoc Deployment of a Script into the Scheduling Queue .....	208
Initiating a Scheduled Deployment of a Script into the Scheduling Queue .....	210
Creating Multiple Deployments for a Scheduled Script .....	210

Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue .....	213
Understanding Scheduled Script Deployment Statuses .....	213
Executing a Scheduled Script in Certain Contexts .....	215
Setting Recovery Points in Scheduled Scripts .....	216
Understanding Memory Usage in Scheduled Scripts .....	216
Monitoring a Scheduled Script's Runtime Status .....	217
Monitoring a Scheduled Script's Governance Limits .....	219
Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus .....	219
Scheduled Script Samples .....	221
<b>Portlet Scripts .....</b>	<b>226</b>
What Are Portlet Scripts? .....	226
Portlet Script Execution .....	227
Assigning the Portlet Preference to a Script Parameter .....	227
Running a Portlet Script in NetSuite .....	227
Displaying Portlet Scripts on the Dashboard .....	228
Portlet Scripts Samples .....	228
<b>Mass Update Scripts .....</b>	<b>233</b>
What Are Mass Update Scripts? .....	233
Mass Update Script Execution .....	235
Running a Mass Update Script in NetSuite .....	236
Mass Update Scripts Samples .....	236
Updating a field that is available through inline edit .....	236
Updating a field that is not available through inline edit .....	237
Updating a field based on a script parameter value .....	237
<b>Bundle Installation Scripts .....</b>	<b>241</b>
What are Bundle Installation Scripts? .....	241
Setting Up a Bundle Installation Script .....	243
Sample Bundle Installation Script .....	246
<b>Setting Runtime Options .....</b>	<b>249</b>
Setting Runtime Options Overview .....	249
Setting Script Execution Event Type from the UI .....	250
Setting Script Execution Log Levels .....	251
Executing Scripts Using a Specific Role .....	252
Setting Available Without Login .....	255
Setting Script Deployment Status .....	257
Defining Script Audience .....	259
<b>Creating Script Parameters (Custom Fields) .....</b>	<b>261</b>
Creating Script Parameters Overview .....	261
Why Create Script Parameters? .....	262
Creating Script Parameters .....	263
Referencing Script Parameters .....	265
Setting Script Parameter Preferences .....	266
<b>Searching with SuiteScript .....</b>	<b>269</b>
Searching Overview .....	269
Understanding SuiteScript Search Objects .....	270
Search Samples .....	273
Creating Saved Searches .....	273
Using Existing Saved Searches .....	274
Filtering a Search .....	275
Returning Specific Fields in a Search .....	278
Searching on Custom Records .....	279
Searching Custom Lists .....	280
Executing Joined Searches .....	280
Searching for an Item ID .....	285

Searching for Duplicate Records .....	285
Performing Global Searches .....	286
Searching CSV Saved Imports .....	286
Using Formulas, Special Functions, and Sorting in Search .....	286
Using Summary Filters in Search .....	287
Supported Search Operators, Summary Types, and Date Filters .....	288
Search Operators .....	288
Search Summary Types .....	289
Search Date Filters .....	290
Working with UI Objects .....	293
UI Objects Overview .....	293
Creating Custom NetSuite Pages with UI Objects .....	295
InlineHTML UI Objects .....	297
Building a NetSuite Assistant with UI Objects .....	298
NetSuite UI Object Assistant Overview .....	298
Understanding NetSuite Assistants .....	298
Using UI Objects to Build an Assistant .....	301
Understanding the Assistant Workflow .....	301
Using Redirection in an Assistant Workflow .....	303
Assistant Components and Concepts .....	303
UI Object Assistant Code Sample .....	306
Debugging SuiteScript .....	315
Working with the SuiteScript Debugger .....	315
SuiteScript Debugger Overview .....	315
Using the SuiteScript Debugger .....	315
Before Using the SuiteScript Debugger .....	317
Ad Hoc Debugging .....	319
Deployed Debugging .....	321
SuiteScript Debugger Interface .....	326
SuiteScript Debugger Buttons .....	326
SuiteScript Debugger Tabs .....	327
SuiteScript Debugger Metering and Permissions .....	331
SuiteScript Debugger Keyboard Shortcuts .....	333
SuiteScript Debugger Glossary .....	334
SuiteScript API .....	335
SuiteScript API Overview .....	335
SuiteScript Functions .....	336
Record APIs .....	345
Subrecord APIs .....	373
Field APIs .....	379
Sublist APIs .....	390
Search APIs .....	416
Scheduling APIs .....	424
Execution Context APIs .....	432
UI Builder APIs .....	436
Application Navigation APIs .....	441
Date APIs .....	452
DateTime Time Zone APIs .....	455
Currency APIs .....	459
Encryption APIs .....	461
XML APIs .....	461
File APIs .....	469
Error Handling APIs .....	473
Communication APIs .....	473

Configuration APIs .....	482
SuiteFlow APIs .....	485
Portlet APIs .....	487
SuiteAnalytics APIs .....	489
User Credentials APIs .....	491
Job Manager APIs .....	492
<b>SuiteScript Objects .....</b>	<b>495</b>
Standard Objects .....	495
nlobjConfiguration .....	496
nlobjContext .....	501
nlobjCredentialBuilder(string, domainString) .....	516
nlobjCSVImport .....	520
nlobjDuplicateJobRequest .....	523
nlobjEmailMerger .....	526
nlobjError .....	530
nlobjFile .....	532
nlobjFuture .....	542
nlobjJobManager .....	543
nlobjLogin .....	544
nlobjMergeResult .....	546
nlobjPivotColumn .....	546
nlobjPivotRow .....	549
nlobjPivotTable .....	551
nlobjPivotTableHandle .....	552
nlobjRecord .....	553
nlobjReportColumn .....	586
nlobjReportColumnHierarchy .....	587
nlobjReportDefinition .....	588
nlobjReportForm .....	594
nlobjReportRowHierarchy .....	594
nlobjRequest .....	595
nlobjResponse .....	600
nlobjSearch .....	609
nlobjSearchColumn(name, join, summary) .....	627
nlobjSearchFilter .....	635
nlobjSearchResult .....	639
nlobjSearchResultSet .....	642
nlobjSelectOption .....	644
nlobjSubrecord .....	646
UI Objects .....	647
nlobjAssistant .....	648
nlobjAssistantStep .....	667
nlobjButton .....	671
nlobjColumn .....	673
nlobjField .....	675
nlobjFieldGroup .....	686
nlobjForm .....	689
nlobjList .....	709
nlobjPortlet .....	714
nlobjSubList .....	722
nlobjTab .....	729
nlobjTemplateRenderer .....	730
SuiteScript API - Alphabetized Index .....	734
<b>SuiteScript Reference .....</b>	<b>739</b>

SuiteScript Reference .....	739
How to Use SuiteScript Records Help .....	739
SuiteScript References Overview .....	740
Working with the SuiteScript Records Browser .....	741
Finding a Record or Subrecord .....	741
Understanding the Record Summary .....	741
SuiteScript Supported Records .....	744
Activities .....	750
Activity .....	750
Event .....	750
Phone Call .....	750
Project Task .....	751
Resource Allocation .....	753
Task .....	755
Work Calendar .....	755
Entities .....	756
Competitor .....	756
Contact .....	756
Customer .....	757
Employee .....	758
Entity .....	759
Generic Resource .....	759
Lead .....	760
Other Name .....	761
Partner .....	761
Project (Job) .....	762
Project Template .....	764
Prospect .....	765
Vendor .....	766
Items .....	768
Using Item Records in SuiteScript .....	768
Assembly Item .....	770
Description .....	770
Discount .....	771
Download Item .....	771
Gift Certificate Item .....	771
Inventory Item .....	771
Item Group .....	772
Item Search .....	772
Kit .....	772
Lot Numbered Assembly Item .....	773
Lot Numbered Inventory Item .....	773
Markup .....	773
Non-Inventory Part .....	773
Other Charge Item .....	774
Payment .....	774
Reallocate Items .....	774
Serialized Assembly Item .....	775
Serialized Inventory Item .....	775
Service .....	775
Shipping Item .....	776
Subtotal .....	776
Communications .....	778
Message .....	778

Note .....	779
Transactions .....	780
Assembly Build .....	781
Assembly Unbuild .....	782
Bin Putaway Worksheet .....	782
Bin Transfer .....	783
Blanket Purchase Order .....	783
Cash Refund .....	784
Cash Sale .....	785
Charge .....	786
Check .....	787
Credit Card Charge .....	787
Credit Card Refund .....	789
Credit Memo .....	790
Customer Deposit .....	790
Customer Payment .....	791
Customer Refund .....	791
Deposit .....	792
Deposit Application .....	794
Estimate / Quote .....	794
Expense Report .....	795
Intercompany Journal Entry .....	795
Intercompany Transfer Order .....	797
Inventory Adjustment .....	798
Inventory Cost Revaluation .....	798
Inventory Count .....	800
Inventory Detail .....	802
Inventory Transfer .....	802
Invoice .....	803
Item Demand Plan .....	803
Item Fulfillment .....	805
Item Receipt .....	806
Item Supply Plan .....	807
Journal Entry .....	808
Landed Cost .....	810
Manufacturing Operation Task .....	812
Manufacturing Planned Time .....	813
Multi-Book Accounting Transaction .....	814
Opportunity .....	815
Order Schedule .....	816
Paycheck .....	816
Paycheck Journal .....	817
Payroll Batch .....	819
Payroll Batch Employee .....	821
Purchase Contract .....	822
Purchase Order .....	823
Requisition .....	823
Return Authorization .....	825
Revenue Arrangement .....	825
Revenue Commitment .....	827
Revenue Commitment Reversal .....	827
Sales Order .....	827
Statistical Journal Entry .....	828
Time .....	831

Transaction Search .....	831
Transfer Order .....	831
Vendor Bill .....	832
Vendor Credit .....	832
Vendor Payment .....	833
Vendor Return Authorization .....	833
Work Order .....	834
Work Order Close .....	834
Work Order Completion .....	835
Work Order Issue .....	836
Support .....	837
Case .....	837
Issue .....	837
Solution .....	837
Task .....	838
Topic .....	838
File Cabinet .....	839
Folder .....	839
Lists .....	840
Account .....	841
Accounting Book .....	841
Accounting Context .....	843
Accounting Period .....	844
Allocation Schedule .....	845
Amortization Schedule .....	846
Amortization Template .....	847
Billing Class .....	849
Billing Rate Card .....	850
Billing Schedule .....	852
Bin .....	854
Class .....	854
Consolidated Exchange Rate .....	854
Currency .....	856
Customer Category .....	857
Department .....	857
Expense Category .....	857
Fair Value Price .....	857
Gift Certificate .....	858
Global Account Mapping .....	858
Group .....	860
Intercompany Allocation Schedule .....	861
Inventory Number .....	863
Item Account Mapping .....	863
Item Revision .....	865
Location .....	865
Manufacturing Cost Template .....	866
Manufacturing Routing .....	866
Nexus .....	866
Payroll Item .....	867
Price Level .....	869
Project Expense Type .....	869
Revenue Recognition Plan .....	870
Revenue Recognition Schedule .....	871
Revenue Recognition Template .....	871

Role .....	872
Sales Tax Item .....	872
Subsidiary .....	873
Tax Control Account .....	873
Tax Group .....	877
Tax Period .....	878
Tax Type .....	879
Term .....	879
Unit of Measure .....	879
Vendor Category .....	880
Workplace .....	880
Customization .....	881
Custom List .....	881
Custom Segment Fields and Values .....	882
Using SuiteScript to Create Values for Existing Custom Segments .....	882
Using SuiteScript to Set Values for Custom Segment Fields .....	883
Custom Transaction .....	884
Scheduled Script Instance .....	886
Script .....	887
Script Deployment .....	890
Marketing .....	894
Campaign .....	894
Campaign Template .....	894
Coupon Code .....	895
Email Template .....	896
Promotion .....	897
Website .....	898
Web Site Setup .....	898
Commerce Category .....	900
Scriptable Sublists .....	902
Access Sublist (contact roles) .....	904
Accounts Sublist .....	904
Accrued Time Sublist .....	904
Adjustments Sublist .....	905
Apply Sublist .....	905
Assignees Sublist .....	905
Attendees Sublist .....	906
Billable Expenses Sublist .....	906
Billable Items Sublist .....	906
Billable Time Sublist .....	907
Bin Numbers Sublist .....	907
Company Contributions Sublist .....	907
Company Taxes Sublist .....	908
Competitors Sublist .....	908
Credits Sublist .....	908
Currencies Sublist .....	909
Custom Child Record Sublists .....	909
Deductions Sublist .....	920
Demand Plan Detail Sublist .....	921
Deposits Sublist .....	926
Direct Mail Sublist .....	927
Download Sublist .....	927
Earnings Sublist .....	927
E-mail Sublist .....	928

Employee Taxes Sublist .....	928
Escalate To Sublist .....	928
Expenses Sublist .....	929
Group Pricing Sublist .....	929
Item Fulfillment/Receipt Sublist .....	929
Items Sublist .....	930
Item Pricing Sublist .....	931
Lead Nurturing Sublist .....	931
Line Sublist .....	931
Members Sublist .....	932
Orders Sublist .....	932
Other Events Sublist .....	932
Partners Sublist .....	933
Pricing Sublist .....	933
Predecessors Sublist .....	948
Related Solutions Sublist .....	948
Resources Sublist .....	948
Sales Team Sublist .....	948
Shipping Sublist .....	949
Site Category .....	949
Time Tracking Sublist .....	950
Topics Sublist .....	950
Units .....	950
Vendors .....	951
Related Items Sublist .....	951
Record Initialization Defaults .....	952
Permission Names and IDs .....	962
Feature Names and IDs .....	977
Preference Names and IDs .....	984
Supported File Types .....	997
Button IDs .....	999
Supported Tasklinks .....	1002
SuiteScript Governance .....	1098
API Governance .....	1098
Script Usage Unit Limits .....	1100
Monitoring Script Usage .....	1101
Governance on Script Logging .....	1102
Search Result Limits .....	1103
Multiple Shipping Routes and SuiteScript .....	1104
Referencing the currencyname Field in SuiteScript .....	1111
SuiteScript Developer Resources .....	1112
SuiteScript Developer Resources .....	1112
SuiteScript and SuiteFlow Impact of Version 2014 Release 2 Address Customization Changes .....	1112
SuiteScript Best Practices .....	1116
General Development Guidelines .....	1116
Suitelets and UI Object Best Practices .....	1118
User Event Best Practices .....	1119
Scheduled Script Best Practices .....	1120
Client Script Best Practices .....	1121
Security Considerations .....	1122
Script Optimization .....	1123

# SuiteScript - The Basics

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

If you are new to SuiteScript, it is recommended that you see these topics in order. You do not need to read every topic that is associated with each Overview, but you should at least read the Overviews to get a sense of how to use SuiteScript in your account.

1. [What is SuiteScript?](#)
2. [Running a Script in NetSuite](#)
3. [SuiteScript API Overview](#)
4. [Script Types Overview](#)
5. [SuiteScript Reference](#)
6. [Setting Up Your SuiteScript Environment](#)

 **Important:** Throughout your SuiteScript development, you may refer to the SuiteBuilder Guide, a user guide that is available in the NetSuite Help Center. This guide provides detailed information on creating custom records, forms, fields, and sublists. In SuiteScript, much of what you will be doing is extending or getting | setting values on many of these custom elements. The SuiteBuilder Guide provides a basic understanding of how these elements are created and their relationship to one another. In the help center, see the help topic [SuiteBuilder Overview](#) to learn more about SuiteBuilder.

## Additional Topics

After you understand the basic concepts behind SuiteScript and how to run a script in NetSuite, see the following topics for details on how to maximize SuiteScript in your account. These topics do not need to be read in order. However, as you progress through SuiteScript development, you will refer to each section often:

- [Working with Records in SuiteScript](#) - Defines what a NetSuite record is as it pertains to SuiteScript. Also provides links to the Record APIs you will use when working with the entire record object.
- [Working with Fields](#) - Defines what a field is as it pertains to SuiteScript. Also provides links to the Field APIs you will use when working with fields on a record.
- [Working with Subtabs and Sublists](#) - Defines what a sublist is as it pertains to SuiteScript. Also provides links to the Sublist APIs you will use when working with sublists on a record.
- [Setting Runtime Options Overview](#) - Provides additional information on the types of runtime options available on a Script Deployment record.
- [Creating Script Parameters Overview](#) - Defines the concept of "script parameters" as they pertain to SuiteScript. Also provides information on how to assign company, user, or portlet preference values to a script parameter.
- [Searching Overview](#) - Explains how to search NetSuite using SuiteScript and the types of searches that are supported in scripting.

- [UI Objects Overview](#) - Explains the concept of UI objects and how these objects can be used in NetSuite to extend your application.
- [SuiteScript Governance](#) - Describes governance limits that are applied to each API and each SuiteScript type.
- [SuiteScript Developer Resources](#) - Provides a central location to access SuiteScript samples, tutorials, and FAQs. Also provided are links to the NetSuite User Group and the Developer Portal.

# What is SuiteScript?

The following topics are covered in this section. If you are new to SuiteScript they should be read in order.

- [What can I do with the SuiteScript API?](#)
- [Using the SuiteScript API with NetSuite Records](#)

## What can I do with the SuiteScript API?



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

SuiteScript is a JavaScript-based API that gives developers the ability to extend NetSuite beyond the capabilities provided through SuiteBuilder point-and-click customization.

The majority of NetSuite forms, records, customization objects and their event/trigger points are programmatically accessible through SuiteScript. What you decide to do with SuiteScript depends on which part of NetSuite you are trying to extend, search, or process.

When you think about using SuiteScript in NetSuite, you must ask yourself:

1. [What do I want to do?](#)
2. [Which APIs support what I want to do?](#)
3. [How do I run a script in NetSuite?](#)

## What do I want to do?

The following are some of the uses for SuiteScript. Next to each use case is a link to the NetSuite script type you might use to programmatically accomplish the tasks involved.

Using SuiteScript you can:

- Perform custom business processing when NetSuite records are updated, created, deleted (using [User Event Scripts](#)).
- Perform custom validations and calculations in the browser client (using [Client Scripts](#)).
- Create custom user interfaces (using script types such as [Suitelets](#) or [User Event Scripts](#) and [UI Objects](#)).
- Run batch processes (using [Scheduled Scripts](#)).
- Execute NetSuite searches (using script types such as [User Event Scripts](#) or [Scheduled Scripts](#)).
- Perform various utility processing such as sending email and faxes, creating and uploading files, or working with XML documents (using script types such as [User Event Scripts](#) or [Suitelets](#)).
- Create custom dashboard portlets (using [Portlet Scripts](#)).
- Perform processing in target accounts for bundled solutions as part of bundle installation or update (using [Bundle Installation Scripts](#)).

## Which APIs support what I want to do?

In the documentation, the SuiteScript API is organized by the types of tasks most developers want to perform. See [SuiteScript API Overview](#) to get started with the SuiteScript API.

See [SuiteScript Functions](#) to see how all APIs are organized. The documentation for each API lists whether the API can be used in client, user event, scheduled, Suitelet, or portlets scripts.

## How do I run a script in NetSuite?

The overall process for getting a script to run in NetSuite is fairly basic. This process includes:

1. Creating a JavaScript file for your script.
2. Uploading the file into NetSuite.
3. Creating a NetSuite “Script” record for your script.
4. Defining script runtime options on a NetSuite Script Deployment page.

For complete details on each step in the process, start with the [Running Scripts in NetSuite Overview](#) topic in the NetSuite Help Center.

## Using the SuiteScript API with NetSuite Records

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The figure below shows a standard Sales Order record in NetSuite.

The figure outlines the basic components of the record, such as:

1. Record object
2. Body fields
3. Buttons and Actions
4. Subtabs
5. Sublists

Many of the APIs in SuiteScript are used to get and set values on each of these components. You can also use SuiteScript to create these components.

## Records

Use [Record APIs](#) to interact with the entire record object.

## Fields

Use [Field APIs](#) to interact with the body fields on the main area of the record. Body fields can also appear on a subtab.

## Buttons

The use of SuiteScript on built-in buttons is not currently supported. However, you can add a new button object to a page using the [nlobjButton](#) UI object.

## Tabs

You can programmatically add fields to NetSuite tabs. You can also add custom tabs using the [nlobjTab](#) UI object.

## Sublists

Use [Sublist APIs](#) to interact with “line item” sublist fields.

# Setting Up Your SuiteScript Environment

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

- Environment Setup Overview
- Configuring NetSuite for SuiteScript
- Working with IDEs Other Than SuiteCloud IDE

## Environment Setup Overview

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Before working with SuiteScript, you should configure both your NetSuite account and your SuiteScript development environment accordingly. See the following sections:

- Configuring NetSuite for SuiteScript
- Setting Up SuiteCloud IDE

## Configuring NetSuite for SuiteScript

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You must complete all of the following tasks to enable SuiteScript in your account and to access the internal record and fields IDs that may be required as parameter values in your SuiteScript code. These tasks include:

1. Enabling SuiteScript
2. Showing Record and Field IDs in Your Account
3. Setting Roles and Permissions for SuiteScript



**Important:** After completing these tasks, see the help topic [Setting Up SuiteCloud IDE](#).

## Enabling SuiteScript

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Before you can run SuiteScript in your NetSuite account, you must enable the SuiteScript feature.

### To enable SuiteScript:

1. Go to Setup > Company > Setup Tasks > Enable Features.

2. Click the **SuiteCloud** tab.
3. Under SuiteScript, click the **Client SuiteScript** or **Server SuiteScript** box (or both, depending on the scripts you want to run).
4. Click **Save**.

**Important:** If Client SuiteScript is enabled, the **Custom Code** tab becomes available on entry and transactions forms (see figure). Here you define which client scripts to associate with the **current form**. For information on attaching client scripts to NetSuite forms, see [Running Scripts in NetSuite Overview](#). For information on entry and transactions forms, see the help topic [Custom Forms](#) in the SuiteBuilder (Customization) Guide.

## Showing Record and Field IDs in Your Account

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

After enabling the SuiteScript feature, NetSuite recommends that you enable the **Show Internal IDs** preference. Enabling this preference lets you see the internal IDs for all fields and records in NetSuite.

When referencing a specific record or field in SuiteScript, you use the internal ID. Even if the record or field's UI label is changed, the internal ID will remain constant.

After enabling **Show Internal IDs**, see these sections for steps on viewing the IDs from within NetSuite:

- [How do I find a record's internal ID?](#)
- [How do I find a field's internal ID?](#)

**Important:** You obtain the internal ID of a record type (for example, 'salesorder') by going to the SuiteScript Records Browser. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

### To show internal NetSuite IDs:

1. Go to Home > Set Preferences.

2. Click the **General** tab and then click the **Show Internal IDs** check box (see figure).
3. Click **Save**.

**Note:** For examples of how internal IDs are referenced in the SuiteScript API, see `nlapiLoadRecord(type, id, initializeValues)` or `nlapiSearchRecord(type, id, filters, columns)`. Also note that when writing SuiteScript, **all record and field IDs must be in lowercase**.

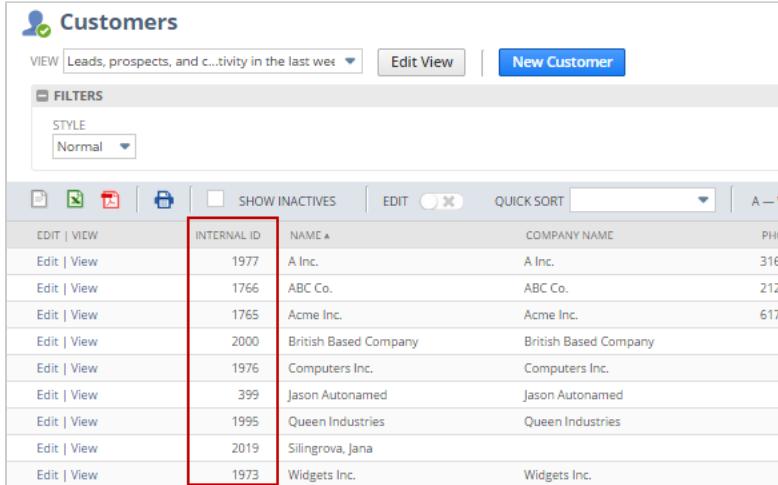
## How do I find a record's internal ID?

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

A record's internal ID is unique and associated with the record at the time it is created. After the **Show Internal IDs** preference is enabled, the internal IDs for each record are displayed in the Internal ID column of record lists (see figure).

For example, to see an internal ID for a specific customer record, go to Lists > Relationships > Customer. In the Internal ID column, the internal ID appears next to each record in the Customers list (see figure).

**Note:** See [Showing Record and Field IDs in Your Account](#) for steps on enabling the **Show Internal IDs** preference.



The screenshot shows the 'Customers' list view in NetSuite. The top navigation bar includes 'VIEW' (set to 'Leads, prospects, and activity in the last week'), 'Edit View', and a 'New Customer' button. Below the view header are 'FILTERS' and 'STYLE' dropdowns ('Normal'). The main area has a toolbar with icons for 'EDIT', 'VIEW', 'CREATE', 'DELETE', 'REFRESH', 'EXPORT', and 'SHOW INACTIVES'. To the right of these are 'EDIT', 'QUICK SORT' (set to 'A - V'), and a search bar. The table has columns: 'INTERNAL ID', 'NAME', 'COMPANY NAME', and 'PHONE'. The 'INTERNAL ID' column is highlighted with a red border. Data rows include: 1977, A Inc., A Inc., 316-; 1766, ABC Co., ABC Co., 212-; 1765, Acme Inc., Acme Inc., 617-; 2000, British Based Company, British Based Company, ; 1976, Computers Inc., Computers Inc., ; 399, Jason Autonamed, Jason Autonamed, ; 1995, Queen Industries, Queen Industries, ; 2019, Silingrova, Jana, ; 1973, Widgets Inc., Widgets Inc., .

If the **Show Internal IDs** preference is NOT enabled, or if the internal IDs are not displayed on a particular page within NetSuite, you can see the internal ID for a record by hovering over a link to that record. The internal ID is displayed as a parameter in the URL in the browser status bar.

The following figure shows that if you hover over a link to the ABC Co. customer record, the internal record ID ( 1766 ) appears in the browser status bar.

The screenshot shows the 'Customers' list page in NetSuite. At the top, there's a 'VIEW' dropdown set to 'Leads, prospects, and activity in the last week', an 'Edit View' button, and a 'New Customer' button. Below that is a 'FILTERS' section with a 'STYLE' dropdown set to 'Normal'. The main area is a table with columns: EDIT | VIEW, NAME ▲, COMPANY NAME, and PHO. The table lists several companies like 'A Inc.', 'ABC Co.', 'Acme Inc.', etc. A red box highlights the URL in the browser address bar at the bottom of the page: <https://system.netsuite.com/app/common/entity/custjob.nl?id=1766>.

**Important:** You obtain the internal ID of a record type (for example, 'salesorder') by going to the [SuiteScript Records Browser](#). For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## How do I find a field's internal ID?

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Internal field IDs must be used when calling a field from a SuiteScript API.

### Internal IDs for Custom Fields on List Pages

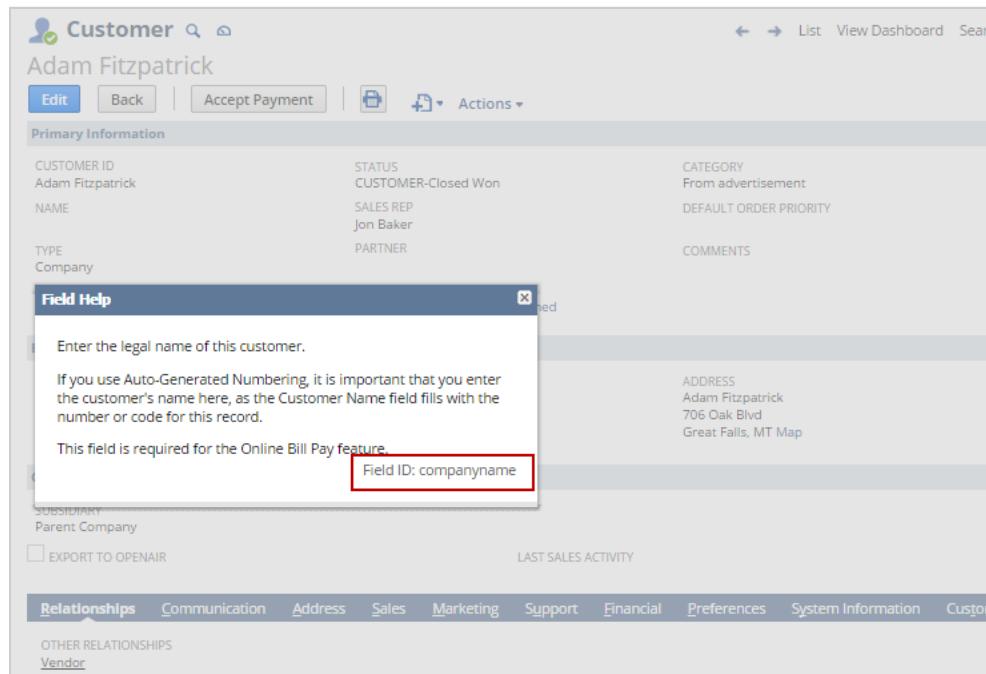
When the **Show Internal IDs** preference is enabled, internal IDs for each custom field are displayed in the Internal ID column of a custom field page. For example, to see the internal IDs for custom CRM fields, go to Customization > Lists, Records, & Fields > CRM Fields. The ID column appears in the list of custom fields, as shown in the figure.

The screenshot shows the 'Custom CRM Fields' list page. At the top, there's a 'New' button. Below that is a 'FILTERS' section with a 'FROM BUNDLE' dropdown. The main area is a table with columns: #, DESCRIPTION, FROM BUNDLE, ID, TYPE, LIST, TAB ▲, and TAB ▼. The table lists three custom fields: 'Days Open' (internal ID: custevent2), 'Inbound Memo' (internal ID: custevent\_inboundmemo), and 'Current Assigned To' (internal ID: custevent4). A red box highlights the 'ID' column.

### Internal IDs for Standard and Custom Fields in Field Level Help Window

With **Show Internal IDs** enabled, you can also view internal field IDs for both standard and custom fields, by clicking the field label in the UI. The figure below shows the Field Level Help window that

opens when a field label is clicked, in this case, the Company label. The internal ID for the Company field is `companynname`, which appears in the bottom-right corner of the Field Level Help window.



**i Note:** When creating custom fields, you can specify your own field ID, or you can accept the default ID assigned by NetSuite. To ensure that the field IDs make sense in the context of your business environment, it is recommended that you define your own custom field IDs. For detailed information on creating custom fields and assigning custom field IDs, refer to [Custom Fields](#) in the NetSuite Help Center.

## Setting Roles and Permissions for SuiteScript

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

NetSuite provides many standard roles with predefined permissions. A role is a set of permissions that lets customers, vendors, partners, and employees access specific areas of your data. Each role grants access at a certain level for each permission.

Access to the SuiteScript feature is also controlled using roles and permissions. When you assign the **SuiteScript** permission to a role, you are allowing the users who have that role to write, upload, and run SuiteScript files in your company's NetSuite account. To assign the SuiteScript permission to a role, a NetSuite administrator must use the following steps:

### To assign the SuiteScript permission to a role:

1. Go to Setup > Users/Roles > Manage Roles.
2. Next, click **Edit** or **Customize** next to the role.

3. On the **Permissions** tab, select the **Setup** subtab.
4. In the **Permissions** dropdown, select **SuiteScript**.
5. Click **Save**.

Note that there are seven standard roles that already have full access to the SuiteScript feature. Users who have the following roles can already write, upload, and run SuiteScript files.

Role	SuiteScript Access Level
Administrator	FULL
Full Access	FULL
Marketing Manager	FULL
Marketing Administrator	FULL
Sales Manager	FULL
Sales Administrator	FULL
Support Administrator	FULL

- When customizing a role to add SuiteScript capabilities, you must also add permission for customizing entry forms and transaction forms.
- Depending on the NetSuite product you subscribe to, not all of the roles listed in the table above may be available to you. Also, in addition to these standard roles there may be custom roles created with the SuiteScript permissions assigned to them.

## Working with IDEs Other Than SuiteCloud IDE



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Although NetSuite recommends that you use SuiteCloud IDE when writing SuiteScript, you can still use other development tools to create SuiteScript. Note, however, without SuiteCloud IDE, you will not be able to automatically upload SuiteScript files into the NetSuite file cabinet.



**Important:** For more information about NetSuite's recommended development environment, see the help topic [Setting Up SuiteCloud IDE](#).

If you choose to use a development tool or IDE other than SuiteCloud IDE, see the following sections:

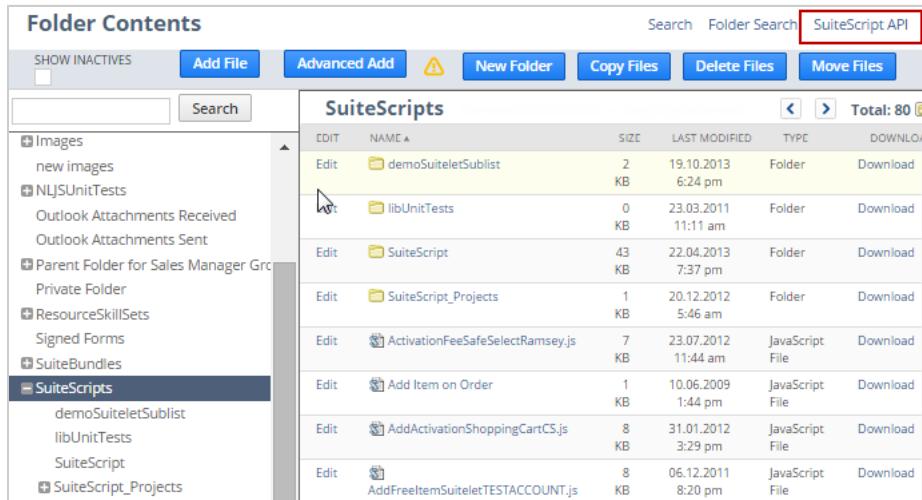
- [Adding the SuiteScript Library File to Your IDE](#)
- [Uploading SuiteScript into the File Cabinet Without the SuiteCloud IDE](#)

## Adding the SuiteScript Library File to Your IDE

If you are working with an IDE other than SuiteCloud IDE, you should still add the SuiteScript library file to your SuiteScript project folder in your IDE.

## To add the SuiteScript library file:

1. In NetSuite, go to Documents > Files > SuiteScripts.
2. Next, click the link to the **SuiteScript API** file (see figure).



The screenshot shows the NetSuite File Cabinet interface. The left sidebar lists various document types like Images, Outlook attachments, and SuiteBundles. The 'SuiteScripts' folder is expanded, showing sub-folders like 'demoSuiteletSublist', 'libUnitTests', 'SuiteScript', and 'SuiteScript\_Projects'. On the right, a grid displays the contents of the 'SuiteScript' folder, including files like 'ActivationFeeSafeSelectRamsey.js' and 'AddActivationShoppingCartCS.js'. A red box highlights the 'SuiteScript API' link at the top right of the grid.

EDIT	NAME	SIZE	LAST MODIFIED	TYPE	DOWNLOAD
Edit	demoSuiteletSublist	2 KB	19.10.2013 6:24 pm	Folder	Download
Edit	libUnitTests	0 KB	23.03.2011 11:11 am	Folder	Download
Edit	SuiteScript	43 KB	22.04.2013 7:37 pm	Folder	Download
Edit	SuiteScript_Projects	1 KB	20.12.2012 5:46 am	Folder	Download
Edit	ActivationFeeSafeSelectRamsey.js	7 KB	23.07.2012 11:44 am	JavaScript File	Download
Edit	Add Item on Order	1 KB	10.06.2009 1:44 pm	JavaScript File	Download
Edit	AddActivationShoppingCartCS.js	8 KB	31.01.2012 3:29 pm	JavaScript File	Download
Edit	AddFreelItemSuiteletTESTACCOUNT.js	8 KB	06.12.2011 8:20 pm	JavaScript File	Download

3. Copy and paste the SuiteScript API file into your IDE.
4. Save the file as a .js file.

## Uploading SuiteScript into the File Cabinet Without the SuiteCloud IDE

After the SuiteScript feature is enabled, a new SuiteScripts folder is created in the NetSuite file cabinet. The file cabinet is considered as the central repository for all your .js SuiteScript files. Therefore, your SuiteScript files should exist in the file cabinet before they can be executed in your NetSuite account.

- The **SuiteScripts** folder within the file cabinet is provided for convenience, however, you can store the script files in any location.
- For steps on enabling the SuiteScript feature, see [Enabling SuiteScript](#).

If you are not uploading your scripts into the file cabinet using the SuiteCloud IDE, use the following steps.

## To upload SuiteScript into the file cabinet:

1. Go to Documents > Files > SuiteScripts.
2. Click the **Add File** button (see figure).

The screenshot shows the NetSuite File Cabinet interface. On the left, there's a tree view of folder structures. A folder named 'SuiteScripts' is expanded, showing its contents: 'demoSuiteletSublist', 'libUnitTests', 'SuiteScript', and 'SuiteScript\_Projects'. At the top of the page, there are several buttons: 'SHOW INACTIVES', 'Add File' (which is highlighted with a red box), 'Advanced Add', 'New Folder', 'Copy Files', 'Delete Files', and 'Move Files'. Above these buttons are links for 'Search', 'Folder Search', and 'SuiteScript API'. The main right-hand area displays a table titled 'SuiteScripts' with columns for 'EDIT', 'NAME', 'SIZE', 'LAST MODIFIED', 'TYPE', and 'DOWNLOAD'. The table lists several files and folders, including 'demoSuiteletSublist' (2 KB, 19.10.2013, Folder, Download), 'libUnitTests' (0 KB, 23.03.2011, Folder, Download), 'SuiteScript' (43 KB, 22.04.2013, Folder, Download), 'SuiteScript\_Projects' (1 KB, 20.12.2012, Folder, Download), 'ActivationFeeSafeSelectRamsey.js' (7 KB, 23.07.2012, JavaScript File, Download), 'Add Item on Order' (1 KB, 10.06.2009, JavaScript File, Download), 'AddActivationShoppingCartCS.js' (8 KB, 31.01.2012, JavaScript File, Download), and 'AddFreelItemSuiteletTESTACCOUNT.js' (8 KB, 06.12.2011, JavaScript File, Download).

- In the File Upload window that appears, navigate to the file you want to upload, select the file, and click **Open**.

**Note:** If you make changes to a SuiteScript file that already exists in the file cabinet, follow steps 1–3 to re-upload the changed file. Click **OK** to overwrite the previous file and load your changes.

# Running Scripts in NetSuite Overview

Running a script in NetSuite includes these basic steps:

- Step 1: Create Your Script
- Step 2: Add Script to NetSuite File Cabinet
- Step 3: Attach Script to Form
- Step 4: Create Script Record
- Step 5: Define Script Deployment



**Important:** Step 3 is for form-level client scripts **only**. If you are creating a user event, scheduled, portlet, Suitelet, or **record** -level client script, skip Step 3, and perform steps 4 and 5.

Step 5 provides the basic steps required for deploying a script into NetSuite. To learn how to specify additional deployment options, see [Setting Runtime Options Overview](#).

# Step 1: Create Your Script

All SuiteScript files must end with a JavaScript (.js) file extension. Although you can use any text editor (including Notepad) to write your SuiteScript .js files, NetSuite recommends you use the SuiteCloud IDE. If you have not installed SuiteCloud IDE, see the help topic [Setting Up SuiteCloud IDE](#) in the NetSuite Help Center.

Depending on what you are trying to do in NetSuite, the code in your .js file can be as basic as a **client** script that never even touches a NetSuite server. It runs purely client-side in the browser and alerts users after they have loaded a specific NetSuite record, for example:

```
function pageInitAlertUser()
{
    alert ('You have loaded a record');
}
```

Alternatively, your script can be as complex as executing a NetSuite search, getting the results, and then transforming the results into a PDF document. See the samples for [nlapiXMLToPDF\(xmlstring\)](#) as an example.

The APIs you use in your code and the logic you write will depend on what you're trying to accomplish in NetSuite. See [What can I do with the SuiteScript API?](#) if you are unsure of what you can do using the SuiteScript API.

After you have created your .js file, see [Step 2: Add Script to NetSuite File Cabinet](#).



**Note:** To see which APIs are included in the SuiteScript API, start with [SuiteScript API Overview](#).

# Step 2: Add Script to NetSuite File Cabinet

If you are writing your script files in SuiteCloud IDE, loading a file into the NetSuite file cabinet is as easy as right-clicking on your file in SuiteCloud IDE and selecting **NetSuite > Upload Selected File(s)**. For more information, see the help topic [Uploading a SuiteScript File](#).

If you have written your .js files in anything other than SuiteCloud IDE, you will need to manually upload your files into NetSuite. See [Uploading SuiteScript into the File Cabinet Without the SuiteCloud IDE](#) for details.



**Note:** The **SuiteScripts** folder in the file cabinet is provided for convenience, however, you can store your script files in any location.

After your script has been added to the NetSuite file cabinet, see either :

- [Step 3: Attach Script to Form](#) (if you want to run a **form-level client script** in NetSuite)
- [Step 4: Create Script Record](#) (if you want to run any other script type. For example, if you want to run a user event, scheduled, portlet, Suitelet, action, or record-level client script, proceed to Step 4.)



# Step 3: Attach Script to Form

Form-level client scripts are “attached” to the forms they run against. Be aware that in NetSuite, there are two different types of client SuiteScript. The information in this section pertains **ONLY** to **form-level** client scripts.



**Important:** For the differences between form- and record-level client scripts, see [Form-level and Record-level Client Scripts](#).

## To attach a form-level client script to a custom form:

1. Ensure that your client script has been uploaded to the File Cabinet. (See [Step 2: Add Script to NetSuite File Cabinet](#).)
2. Navigate to the appropriate custom form in NetSuite. Form-level client scripts can only be attached to custom entry forms, custom transaction forms, and custom online forms. Click Customization > Forms >*[Form]*.
3. Click **Edit** next to the desired custom form, or click **Customize** next to an existing standard form to create a new custom form that is based on the standard version.



**Note:** For more information on creating custom entry, transaction, and online forms, refer to the *SuiteBuilder (Customization) Guide*.

4. On the form’s **Custom Code** subtab, use the **Script File** dropdown list to select your SuiteScript version 1.0 file. The page updates and populates the SuiteScript API Version field. The system also adds several text fields to the page.
5. Use the text fields to enter the functions defined in your script that should be called for each appropriate client event. If you are unsure of which actions trigger each client event, see [Client Event Types](#). To learn how many functions you can execute on one form, see [How Many Client Events Can I Execute on One Form?](#)

The following figure shows a custom sales order form. This form is set as the preferred form for sales orders. What this means is that all customizations made to this form, and any client script file attached to the form, run whenever a NetSuite user navigates to and loads the sales order record. As shown in the screenshot, when a sales order loads, three functions execute:

- The `savRecUpdatePrice` function will execute when the record is saved.
- The `valFieldItemPrice` function will execute when a particular field on the sales order is changed.
- The `recalcTotalAndTax` function will execute when a line item as been added to a sublist.



**Important:** You must enter function names **exactly** as they appear in your script. However, do not include parenthesis or parameter values when you enter the function name on the custom form.

6. If appropriate, use the **Library Script File** dropdown list to select a library file to associate with the form. The library script file should contain any commonly used functions. The file named in the **Script File** field should contain functions specific to the current form.

After you have attached your form-level client script to a form, your script will execute whenever the triggering action occurs. For a list of possible client event triggers, see [Client Event Types](#).

If you have created a form-level client script, you do not have to complete the procedures described in [Step 4: Create Script Record](#) or [Step 5: Define Script Deployment](#).

# Step 4: Create Script Record

After writing your SuiteScript .js file and uploading the file to the file cabinet, you must then create a Script record for the file (see figure).

On the **Script** record you will:

- Add your SuiteScript .js file.
- Define the script owner.
- If applicable, add one or more library files.
- Define the function(s) from your SuiteScript file you want executed.
- If applicable, create script parameters (custom fields) that are unique to the Script record.
- Specify who should be contacted if an error is thrown in your script.

Although you do not need to set every field on the Script record, **at a minimum** you must set the following (see figure):

1. Provide a name for the Script record.
2. Specify the script owner.
3. Load your SuiteScript .js file.
4. Specify the main executing function within the file.

**Important:** See [Steps for Creating a Script Record](#) for more detailed information.

The screenshot shows the 'Script' record creation page in NetSuite. The 'NAME' field (containing 'Sending follow up email') and the 'OWNER' dropdown (containing 'K Wolfe') are highlighted with red boxes. The 'SCRIPT FILE' dropdown (containing 'followupEmail.js') and the 'AFTER SUBMIT FUNCTION' field (containing 'sendNotification') are also highlighted with red boxes. The 'Save' button at the bottom left is also highlighted with a red box.

# Steps for Creating a Script Record

The following steps provide details for creating a Script record. For an overview that explains the purpose of the Script record, be sure to see [Step 4: Create Script Record](#).

## To create a script record:

1. Go to Customization > Scripting > Scripts > New.
- Note that after creating your script record, you can later access the record by going to Customization > Scripting > Scripts to see a list view of all Script records.
2. In the **Script File** field, enter a name for the script record and then click **Create Script Record**.
  3. Select the script type (see figure).

Select Type	
TYPE	DESCRIPTION
Suitelet	Build interactive Web applications by scripting web requests
RESTlet	Build custom RESTful web services
User Event	Define business logic that is triggered when records are created, updated, viewed, or deleted
Scheduled	Schedule complex batch operations or queue them on-demand for execution
Client	Define business logic and perform client-side validation on your forms
Portlet	Publish scriptable portlets to your dashboards and centers
Mass Update	Perform an update to a record as part of a mass update
Workflow Action	Defines a custom action on a record that can be used as part of a workflow
Bundle Installation	Define scripts that run as part of bundle installation or update



**Note:** The Client scripts listed here are record-level client script. These scripts run in addition to any form-level client scripts that might have already been attached to an existing form. For information on the differences between form- and record-level client scripts, see [Form-level and Record-level Client Scripts](#) .

4. In the Script record **Name** field, enter a name for the script record.

You can have multiple deployments of the same SuiteScript file. Therefore, be sure that the name of the Script record is generic enough to be relevant for all deployments.

For example, you may want your SuiteScript (.js) file to execute whenever Vendor records are saved. You might also want this script to execute whenever Customer records are saved. You will need to define two different deployments for the script. However, both deployments will reference the same script / Script record. (Information on defining script deployments is covered in [Step 5: Define Script Deployment](#).)

5. In the **ID** field, if desired, enter a custom ID for the script record. If the **ID** field is left blank, a system-generated internal ID is created for you.

For information on whether you should create your own custom ID, see [Creating a Custom Script Record ID](#).

6. In the **Description** field, if desired, enter a description for the script.
7. In the **Owner** field, select a script owner.

By default the owner is set to the currently logged-in user. Once the Script record is saved, only the owner of the record or a system administrator can modify the record.

8. (Optional) Select the **Inactive** check box if you do not want to deploy the script. When a script is set to Inactive, all of the deployments associated with the script are also inactive. If you wish

to inactivate a specific deployment rather than all deployments of this scripts, go to the Script Deployments page.

9. On the **Scripts** tab, set the following:

- In the **Script File** field, select the SuiteScript .js file to associate with the current script record.

If you have uploaded your script into the NetSuite file cabinet, your script appears in the **Script File** drop-down list. For directions on uploading scripts into the file cabinet, see either of the following sections:

- [Uploading a SuiteScript File](#) - If you are uploading scripts using the SuiteCloud IDE.
- [Uploading SuiteScript into the File Cabinet Without the SuiteCloud IDE](#) - If you are not uploading your scripts using SuiteCloud IDE.

**Note:** If you are maintaining your SuiteScript files outside of the file cabinet, click the + button next to the **Script File** drop-down. In the popup window that appears, browse for your .js file.

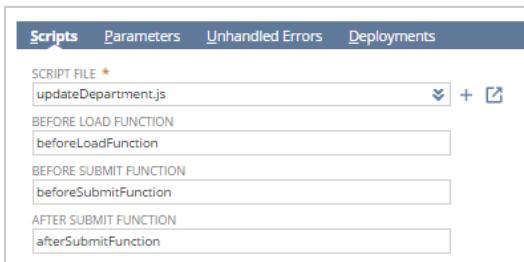
- (Optional) In the **Library Script File** field, select the library files you want to associate with the Script record.

A library script file should contain any commonly used functions, whereas the SuiteScript file should contain functions specific to the current Script record. Note that multiple library files can be added to a script record.

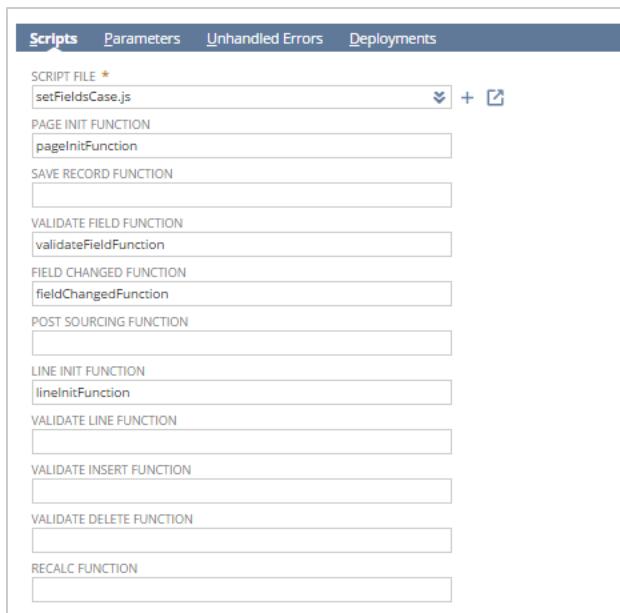
The upload order of your library files only matters if you have two functions with the same name. When there is a function name conflict in a library file, the script will execute the function in the library file that was loaded last.

- In the **Function** field(s), type the name of the function(s) you want executed in the .js file. Do not include the function parentheses or any parameters. For example, type **myFunction** rather than **myFunction(param1, param2)**.
- If defining a User Event script, you can execute one function per operation type. For example, you can have a before load function and an after submit function defined within the same script execution. These functions must exist in either the library script file or the SuiteScript file associated with the script record.

**Note:** For details on the before load, before submit, and after submit operations, see [User Event beforeLoad Operations](#) and [User Event beforeSubmit and afterSubmit Operations](#).



- If defining a record-level Client script, type the names of the functions you want executed when the script runs. As the following figure shows, enter the function name in the field next to the client event that will trigger the function. Note that your functions must exist in either the library script file or the SuiteScript file associated with the script record. You have the option of calling up to eight functions from within the same script file.



- If defining a bundle installation script, type the names of the functions you want executed before the bundle is installed, after the bundle is installed, before the bundle is updated, or after the bundle is updated. Enter the function name in the field next to the bundle deployment event that will trigger the function. Note that these functions must exist in the SuiteScript file associated with the script record. If these functions call functions in other script files, these files should be listed as library files.
10. On the **Parameters** tab, define possible parameters (custom fields) to pass to the functions specified in the previous step.
  11. On the **Unhandled Errors** subtab, define which individual(s) will be notified if script errors occur.

Three types of error notifications are sent:

- An initial email is sent about the first occurrence of an error within an hour.
- An email with aggregated error information for every hour is sent. (The error counter is reset when this email is sent.)
- An email is sent about the first 100 errors that have occurred after the error counter is set to 0.

For example, an error is thrown 130 times within an hour. An initial email is sent. After the 100th occurrence, another email is sent. Since there are an additional 30 occurrences within the same hour, a final summary email is sent at the end of the hour. During the second hour, if there are only 50 occurrences of the error, only one summary email is sent at the end of that hour.



**Note:** By default the **Notify Script Owner** check box is selected.

- (Optional) Select **Notify All Admins** if all admins should be notified.
- (Optional) Select the groups that should be notified. Only existing groups are available in the **Groups** notification drop-down list. To define new groups, go to Lists > Relationships > Groups > New.
- (Optional) Enter the email address of anyone who should be notified. You can also enter a comma-separated list of email addresses.

The screenshot shows the 'Script' creation interface. At the top, there are 'Save' and 'Cancel' buttons. Below that, the 'TYPE' is set to 'User Event'. The 'NAME' field contains 'Sending follow up email'. The 'ID' field contains '\_ue\_follow\_up\_email'. The 'Scripts' tab is selected, followed by 'Parameters', 'Unhandled Errors' (which is underlined in blue, indicating it's active), and 'Deployments'. Under 'Unhandled Errors', there are checkboxes for 'NOTIFY CURRENT USER', 'NOTIFY SCRIPT OWNER' (which is checked), 'NOTIFY ALL ADMINS', and 'NOTIFY EMAILS'. A dropdown menu labeled 'NOTIFY GROUP' is open. The 'NOTIFY EMAILS' field is empty.

**12. From the Save button:**

1. If you want to save the script record, but you are not ready to deploy the script, select the **Deployments** tab, clear the **Deployed** check box, and click **Save**.



**Important:** Scripts do not execute until they are deployed.

2. If you want to save the script record and deploy the script, but you are not yet ready to define the script's runtime/deployment behaviors, click **Save**.
3. If you want to save the script record and automatically open the Script Deployment page, click **Save and Deploy**. Use the Script Deployment page to define runtime behaviors such as when the script will run and which accounts the script will run in.

**13. Now that you have created a Script record for your script, go to Step 5: Define Script Deployment.**



**Note:** Although the Script record has a **Deployments** tab where you can define many of the same deployment options found on the Script Deployment page, it is recommended that you define your deployments on the Script Deployment page. This page provides deployment settings that are not available on the **Deployments** tab of the Script record.

## Creating a Custom Script Record ID

All Script records have an ID. Many SuiteScript APIs contain parameters such as **ID** or **scriptId**. Through these parameters you pass the **scriptId** or **internalId** of the script record. The **scriptId** is considered to be a custom ID you create yourself for the Script record. If you do not create your own **scriptId**, then the system generates an ID for you. In the documentation, the system-generated ID is referred to as the Script record's **internalId**.

The screenshot shows the 'Script' creation interface. At the top, there are 'Save' and 'Cancel' buttons. Below that, the 'TYPE' is set to 'User Event'. The 'NAME' field contains 'Sending follow up email'. The 'ID' field contains '\_ue\_follow\_up\_email'. The 'ID' field is highlighted with a red border. The 'Scripts' tab is selected, followed by 'Parameters', 'Unhandled Errors' (underlined in blue), and 'Deployments'.

For an example of how Script record IDs are used in a SuiteScript API call, see [nlapiScheduleScript\(scriptId, deployId, params\)](#).

**Note:** You can programmatically get the value of a `scriptId` by calling `nlobjContext.getScriptId()`.

If you choose, you can create a custom ID for your **Script** record. If the ID field is left blank on the **Script** record, a system-generated ID is created for you. This is the ID that appears in the ID field once the **Script** record is saved.

Whether creating a custom ID or accepting a system-generated ID, once the script record is saved, the system automatically adds `customscript` to the front of the ID.

## Why Should I Create a Custom ID?

Custom IDs are recommended if you plan to use the **SuiteBundler** feature to bundle the script and deploy it into another NetSuite account. Custom IDs reduce the risk of naming conflicts for scripts deployed into other accounts. (For details on bundling scripts, see the *SuiteBundler Overview* topic in the NetSuite Help Center.)

When creating a custom ID it is recommended that you insert an underscore ( `_` ) before the ID to enhance readability. For example, a custom script ID called `_employeeupdates` will appear as `customscript_employeeupdates` once the **Script** record is saved. Similarly, a custom deployment ID will appear as `customdeploy_employeeupdates` once the **Script Deployment** page is saved.

**Important:** Custom IDs must be in **lowercase** and contain no spaces. Also, custom IDs cannot exceed 30 characters in length. These 30 characters do not include the `customscript` or `customdeploy` prefixes that are automatically appended to the ID.

## Can I Edit an ID?

Although not recommended, you can edit both custom and system-generated IDs once the **Script** record or script deployment is saved. To edit an ID, click the **Change ID** button that appears on both **script record** and **script deployment** pages AFTER each has already been saved.

The following figure shows the **Change ID** button on a **Script Deployment** page after the deployment has been saved.

The screenshot shows a 'Script' deployment page. At the top, there are buttons for 'Save' (highlighted in blue), 'Cancel', 'Reset', and 'Change ID'. The 'Change ID' button is circled in red. Below these buttons is a section labeled 'TYPE' with 'User Event' selected. The 'NAME' field contains 'Sending follow up email'. The 'ID' field contains 'customscript\_ue\_follow\_up\_email'. To the right of the name and ID fields are sections for 'DESCRIPTION', 'OWNER' (set to 'K Wolfe'), and a checkbox for 'INACTIVE' which is unchecked. At the bottom right of the page is a 'Actions' dropdown menu.

After clicking the **Change ID** button, the **Change Script ID** page appears. This page shows the old ID and provides a field for creating a new ID.



**Important:** Once you change a script record or script deployment ID, you **MUST** update all references to that ID in your code files.

# Step 5: Define Script Deployment

After you have created a Script record for your SuiteScript file, you must then “deploy” the script into NetSuite. A script’s deployment definitions, as set on the Script Deployment page, affect its runtime behaviors when it is released into NetSuite.

Some of these deployment definitions include:

- When the script will be executed
- Audience and role restrictions for a script
- Script log levels
- Deployment-specific parameter defaults
- Specific records the script will run against

Note that Script Deployment pages look different for each script type. For example, the Script Deployment page for a user event script will not include an area for you to define the script’s deployment schedule. The Script Deployment page for a scheduled script, however, will include an area for this. Deployment pages for Suitelets will include a field for setting whether the Suitelet executes on a GET or POST request. The deployment page for a global client script will not include such a field.

Because Script Deployment pages vary depending on the script type, see [Steps for Defining a Script Deployment](#) for general steps that are applicable to most script types. See [Setting Runtime Options Overview](#) for information on setting more advanced runtime options. In many cases, these more advanced options are specific to a particular script type.

## Important Things to Note:

- You cannot edit a Script Deployment record while the script associated with the deployment is running in NetSuite.
- Multiple deployments can be applied to the same record. These deployments are executed in the order specified in the UI. If an error occurs in one deployment, subsequent deployed scripts may NOT be executed. When troubleshooting, verify you are executing only one script per record type.

## Steps for Defining a Script Deployment

For an overview of the Script Deployment record, be sure to see [Step 5: Define Script Deployment](#). This section describes why a Script Deployment record is required for each script.

### To define a script deployment:

1. When you save your Script record, you can immediately create a Script Deployment record by selecting **Save and Deploy** from the Script record **Save** button.  
If you want to update a deployment that already exists, go to Customization > Scripting > Script Deployments > [deployment] > Edit.
2. On the Script Deployment page:
  - For Suitelet, Scheduled, and Portlet scripts, in the **Title** field, provide a name for the deployment.

- For User Event and Client scripts, in the **Applies To** field, select the record the script will run against. In the **Applies To** field you can also select **All Records** to deploy the script to all records that officially support SuiteScript. (For a list of these records, see [SuiteScript Supported Records](#).)
3. In the **ID** field, if desired, enter a custom scriptId for the deployment. If you do not create a scriptId, a system-generated internalId is created for you.
- For information on whether to create a custom ID, see [Creating a Custom Script Deployment ID](#).
4. (Optional) Clear the **Deployed** check box if you do not want to deploy the script. Otherwise, accept the default. A script will not run in NetSuite until the **Deployed** check box is selected.
  5. In the **Status** field, set the script deployment status. See [Setting Script Deployment Status](#).
  6. (Optional) In the **Event Type** drop-down list, specify an event type for the script execution. See [Setting Script Execution Event Type from the UI](#).
  7. (Optional) In the **Log Level** field, specify which log messages will appear on the **Execution Log** tab after the script is executed. See [Setting Script Execution Log Levels](#).
  8. In the **Execute as Role** field, select whether you want the script to execute using Administrator privileges, regardless of the permissions of the currently logged in user. See [Executing Scripts Using a Specific Role](#).
  9. On the **Audience** tab, specify the audiences for the script. See [Defining Script Audience](#).
  10. On the **Links** tab (for Suitelets only), if you want to launch your Suitelet from the UI, create a menu link for the Suitelet. See [Running a Suitelet in NetSuite](#).
  11. (Optional) On the **Execution Log** tab, create custom views for all script logging details. See [Creating Script Execution Logs](#).
  12. Click **Save**.

Note that for portlet scripts, you must enable the portlet to display on your dashboard (see [Displaying Portlet Scripts on the Dashboard](#)).

## Creating a Custom Script Deployment ID

Script deployment IDs are necessary for SuiteScript development. Many SuiteScript API calls contain parameters such as *ID*, *scriptId*, and *deployId* that reference the IDs on the **Script Deployment** page.

These parameters allow you to pass the values of an *internalId* (a system-generated ID) or a *scriptId* (a custom ID that you provide). For an example of how script record and script deployment IDs are used in a SuiteScript API call, see [nlapiScheduleScript\(scriptId, deployId, params\)](#).

If you choose, you can create a custom ID for your script deployment. If the **ID** field is left blank on the **Script Deployment** page, a system-generated ID is created for you. This is the ID that appears in the **ID** field after the **Script Deployment** page is saved.

Whether creating a custom ID or accepting a system-generated ID, after the script deployment is saved, the system automatically adds **customdeploy** to the front of the ID.

The following figure shows a list of script deployments (Setup > Customization > Script Deployments). Note that there is a combination of custom IDs (for example, `customdeploy_campaign_assistant`) and system-generated deployment IDs (for example `customdeploy1`). Although `customdeploy1` is the ID for many script deployments, be aware that deployment IDs are unique only within a specific script definition.

Script Deployments			
FILTERS			
<input type="checkbox"/> SHOW UNDEPLOYED			
INTERNAL ID	EDIT   VIEW	ID	SCRIPT ▲
133	Edit   View	customdeploy1	142683_SS_JT
448	Edit   View	customdeploy_advpromo_os_cust_addid_ss	Add Customer Id SL
430	Edit   View	customdeploy_advpromo_add_customerid_ss	Add Customer Id
451	Edit   View	customdeploy_advpromo_add_customer_ss	Add Customer Sa

If you are unsure whether to create your own custom ID or accept a system-generated ID, see [Why Should I Create a Custom ID?](#) for more information.

Also see [Can I Edit an ID?](#) for information on editing IDs.

# Viewing Script Deployments



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

There are several ways to view your script deployments:

- Go directly to the script deployment by clicking Setup > Customization > Scripting > Script Deployments.
- View deployed scripts by clicking View Deployments in the upper-right corner of the Script record.
- Click the Deployments tab on a Script record to see the deployments specific to that Script record. Next, click on a specific deployment to go to the deployment record.

**Remember:** In each specific deployment record you can define default parameter values for **that** deployment. Note that first you must create the script parameter before you can define its value on a Script Deployment record.

For more information, see [Creating Script Parameters Overview](#) in the NetSuite Help Center. Also see [Setting Script Parameter Preferences](#) for information that is specific to setting script parameter values on the Script Deployment record.

- View a list of records that have scripts associated with them at Setup > Customization > Scripting > Scripted Records. For complete details, see [Using the Scripted Records Page](#) in the NetSuite Help Center.

By default, the Scripted Records list displays only those records that have at least one script associated with them.



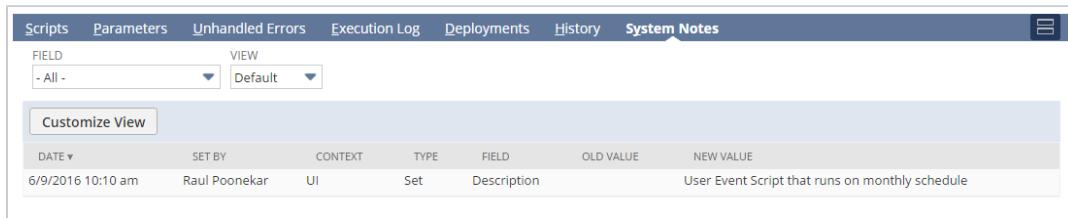
# Viewing Script and Deployment System Notes

To view the activity of your script and deployment, use the System Notes subtab.

## To view script and deployment system notes:

1. Choose an option:
  - Go to Customization > Scripting > Scripts. Click **Edit** beside the script that you want to view.
  - Go to Customization > Scripting > Script Deployments. Click **Edit** beside the deployment that you want to view.
2. Click the **System Notes** subtab.

**i Note:** Previously, script and deployment information was listed on the History subtab, but that subtab is no longer updated. New script and deployment activity is captured on the System Notes subtab.



The screenshot shows a user interface for viewing system notes. At the top, there is a navigation bar with tabs: Scripts, Parameters, Unhandled Errors, Execution Log, Deployments, History, and System Notes. The System Notes tab is currently selected. Below the navigation bar, there is a search and filter section with dropdown menus for 'FIELD' (set to 'All') and 'VIEW' (set to 'Default'). A 'Customize View' button is also present. The main area displays a table of changes. The columns are labeled: DATE, SET BY, CONTEXT, TYPE, FIELD, OLD VALUE, and NEW VALUE. One row in the table is visible, showing a change made on 6/9/2016 at 10:10 am by Raul Poonekar via the UI, where a field named 'Description' was set from 'User Event Script' to 'Script that runs on monthly schedule'.

DATE	SET BY	CONTEXT	TYPE	FIELD	OLD VALUE	NEW VALUE
6/9/2016 10:10 am	Raul Poonekar	UI	Set	Description	User Event Script	Script that runs on monthly schedule

The system note for a change on a script or deployment record captures the following information:

- Date when the change was made
- Who made the change
- Context for the change (for example, UI)
- Type of change, for example, Edit
- Field changed
- Old value
- New value

# Using the Scripted Records Page

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Within NetSuite you can view a list of all records that have user event or global client scripts associated with the record (see figure). By default, only the records that have at least one script associated with them will appear in the list.

**Note:** To see a list of all records in your account, click the Show Undeployed check box in the lower-left corner of the Scripted Records list. You can also use the Script filter drop-down to show only those records associated with specific scripts.

The Scripted Records list is helpful if you are trying to determine whether a record has a script associated with it that might be causing a problem. You can also use this list to drill down to a specific record to specify the execution order of scripts associated with each record, edit script deployment statuses, and deactivate specific deployments.

To view this list, go to Customization > Scripted Records.

Scripted Records				
FILTERS				
EDIT   VIEW	RECORD	SCRIPT ID	SCRIPTS	WORKFLOWS
Edit   View	Campaign	campaign	9	0
Edit   View	Case	supportcase	11	0
Edit   View	Cash Refund	cashrefund	10	0
Edit   View	Cash Sale	cashsale	12	0
Edit   View	Check	check	10	0
Edit   View	Contact	contact	10	0
Edit   View	Credit Memo	creditmemo	10	0
Edit   View	Customer	customer	17	2
Edit   View	Discount Item	discountitem	8	0
Edit   View	Employee	employee	9	0

To change the deployment status of a script associated with a specific record, click **Edit** in the Scripted Records list.

The following figure shows a Scripted Record page for the Case (supportcase) record.

Scripted Record							
<b>Save</b>		<b>Cancel</b>	<b>Reset</b>	More			
NAME	Case	ID	supportcase				
<b>User Event Scripts</b> • <b>Client Scripts</b> • <b>Custom Forms</b> • <b>Workflows</b> <small>You can specify the order in which user event scripts are executed for this scripted record. Note that deploying too many user event scripts of the same trigger type may impact performance. NetSuite recommends deploying no more than 10 scripts of the same trigger type, however, all scripts set to Deployed will execute.</small>							
<b>Move To Top</b>		<b>Move To Bottom</b>					
SCRIPT	STATUS	DEPLOYED	BEFORE LOAD FUNCTION	BEFORE SUBMIT FUNCTION	AFTER SUBMIT FUNCTION	OPTIONS	
SuiteSocial Tab	Released	<input checked="" type="checkbox"/>	newsfeedTab				
SuiteSocial Auto Post	Released	<input checked="" type="checkbox"/>			autoPost		
ClosedCase	Released	<input checked="" type="checkbox"/>			sendCaseEmail		
setFieldOnCase	Released	<input checked="" type="checkbox"/>		setInboundMemo			
Test Attachments and Email	Testing	<input type="checkbox"/>	beforeLoadSendTestEmail				
setEmailReplyCase	Testing	<input type="checkbox"/>		setEmailForm			

On the Scripted Record page, use the **User Event Scripts** tab to:

- Change the script execution order of user event scripts - in other words, have the third script execute first by moving the script to the top of the list
- Change script deployment statuses from Testing to Released
- Set scripts to deployed or undeployed (by checking the Deployed check box)
- View the names of the functions that are set to execute on before load, before submit, and after submit user events

Use the **Client Scripts** tab to:

- Change the script execution order of global client scripts
- Change script deployment statuses from Testing to Released
- Set scripts to deployed or undeployed (by checking the Deployed check box)
- View the names of the functions that are set to execute on various client event triggers

Use the **Custom Forms** tab to:

- View all forms associated with this record type. Even forms that do not have a script attached will appear in this list.
- Access each form directly by clicking the form links
- View the name of the SuiteScript .js file that has been attached to each form
- View the names of the functions that are set to execute on various client event triggers

# Creating Script Execution Logs

During script execution, a detailed script execution log is generated when either an unexpected error occurs or the nlapiLogExecution method is called.

For example, the following Suitelet code generates an execution log that indicates the request type of the Suitelet:

```
nlapiLogExecution('DEBUG', 'Suitelet Details', 'Suitelet method = ' + request.getMethod());
```

For more information about the nlapiLogExecution function, see [nlapiLogExecution\(type, title, details\)](#).

There are two ways to view the script execution logs:

- The **Script Execution Logs** list that can be accessed through Customization > Scripting > Script Execution Logs.  
The list of script execution logs is an enhanced repository that stores all log details for 30 days. The filter options on this page allow you to search for specific logs. See [Viewing a List of Script Execution Logs](#).
- The **Execution Log** tab that appears on Script pages, Script Deployment pages, and the SuiteScript Debugger.  
The execution log tab displays logs for a specific script, but these logs are not guaranteed to persist for 30 days. These logs are searchable, and you can customize views to find specific logs. See [Using the Script Execution Log Tab](#).

## Viewing a List of Script Execution Logs



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The script execution log list is an enhanced repository that stores log details across all scripts for 30 days. The list can be accessed through Customization > Scripting > Script Execution Logs.

On this page, you can perform the following tasks:

- Search for specific logs using filter options, such as log level, execution date range, and script name.
- Download the list as a CSV file or an Excel spreadsheet.
- Print the list.

The following figure shows a script execution log list that is filtered to display DEBUG level logs created on a particular week for a particular script:

Script Execution log							
FILTERS		LOG LEVEL	SCRIPT	DEPLOYMENT ID	USER	TITLE	DETAIL
DATE/TIME ▾	DATE	FROM	TO	SCRIPT	DEPLOYMENT ID	TOTAL: 4	
7/22/2015 3:29:33 pm	DEBUG	Financial Report Systems Notes	CUSTOMDEPLOY_FRS_FORMSUITELET_NOTES	45 Wolfe, K	Record type of current record		
7/22/2015 5:10:31 pm	DEBUG	Financial Report Systems Notes	CUSTOMDEPLOY_FRS_FORMSUITELET_NOTES	45 Wolfe, K	Suitelet Details	Suitelet method = GET	
7/23/2015 5:05:38 pm	DEBUG	Financial Report Systems Notes	CUSTOMDEPLOY_FRS_FORMSUITELET_NOTES	45 Wolfe, K	Suitelet Details	Suitelet method = GET	
7/23/2015 5:05:38 pm	DEBUG	Financial Report Systems Notes	CUSTOMDEPLOY_FRS_FORMSUITELET_NOTES	45 Wolfe, K	Suitelet Details	The note was successfully stored.	

# Using the Script Execution Log Tab



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Script execution details are logged on the **Execution Log** tab that appears on the Script page, Script Deployment page, and SuiteScript Debugger. These logs are searchable, and you can customize views to find specific logs.



**Important:** When using the SuiteScript Debugger to debug a script, all logging details appear on the Execution Log tab in the Debugger. To have logging details appear on the Execution Log tab of the Script Deployment page, you must deploy the script first.

The following figure shows two types of execution logs for a Suitelet. One is an unexpected error that is generated because a method was not defined in a Suitelet script. The other is a user-generated execution log that is generated by the following line in the Suitelet code:

```
nlapiLogExecution('DEBUG', 'Suitelet Details', 'Suitelet method = ' + request.getMethod());
```

VIEW	TYPE	TITLE	DATE	TIME	USER	DETAILS	REMOVE
View	System	UNEXPECTED_ERROR	7/22/2015	10:10 am	45 Wolfe, K	ReferenceError: "processSuitelet" is not defined. (frs_notes.js\$5178#26)	<a href="#">Remove</a>
View	Debug	Suitelet Details	7/22/2015	10:10 am	45 Wolfe, K	Suitelet method = GET	<a href="#">Remove</a>

By default, the View drop-down field is set to **Default Script Notes View**. This default view shows all log types from the **current** day's script executions.

To see script executions for days other than the current day, click the **Customize View** button. Specify search criteria such as script execution dates, details, names, and script types (see figure), and then name your custom view in the **Search Title** field.

The following figure shows a customized view of the execution log. The new view type has been selected from the View drop-down. This view shows log details for days other than the current day.

The screenshot shows the 'Execution Log' tab selected in the top navigation bar. A dropdown menu shows 'Debug' selected under 'TYPE'. The 'VIEW' dropdown shows 'Custom Default Script Notes View'. Below the header are buttons for 'Customize View', 'Remove All', and 'Refresh'. The main area is a table with columns: VIEW, TYPE, TITLE, DATE, TIME, USER, DETAILS, and REMOVE. The 'DATE' column is highlighted with a red border. The table contains the following data:

VIEW	TYPE	TITLE	DATE	TIME	USER	DETAILS	REMOVE
View	Debug	Suitelet Details	7/23/2015	10:05 am	45 Wolfe, K	The note was successfully stored.	Remove
View	Debug	Suitelet Details	7/23/2015	10:05 am	45 Wolfe, K	Suitelet method = GET	Remove
View	Debug	Record type of current record	7/22/2015	7:53 am	45 Wolfe, K		Remove

 **Important:** NetSuite purges system errors older than **60 days** and user-generated logs older than **30 days** on a daily basis. Because log persistence is not guaranteed, NetSuite recommends using custom records if you want to store script execution logs for extended periods. See [Governance on Script Logging](#).

# Using the Script Queue Monitor

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The Script Queue Monitor (Beta) SuiteApp is intended for use in accounts with one or more SuiteCloud Plus licenses. SuiteCloud Plus allows larger accounts to divide their scheduled script work so that scripts can be processed more efficiently. Generally, companies that run NetSuite are provided a single queue for running their scheduled scripts. Companies can upgrade their number of scheduled script queues from one to five with the purchase of a SuiteCloud Plus license. The purchase of two SuiteCloud Plus licenses provides 10 queues and the purchase of three licenses provides 15 queues. For more information about SuiteCloud Plus, see [Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus](#) and [Using SuiteCloud Plus](#).

The Script Queue Monitor provides charts and a list of status details for scheduled script instances running on multiple queues. Account administrators can use the visualizations provided by the Script Queue Monitor to review and manage script queue usage. This information can inform the retargeting of scheduled script deployments to different queues, to maximize the benefit obtained from SuiteCloud Plus.

**Note:** The Script Queue Monitor is currently considered a beta feature.

The Script Queue Monitor currently is not supported if you are running NetSuite in Internet Explorer 11.

For more information, see the following:

- [Installing and Accessing the Script Queue Monitor](#)
- [Script Queue Monitor User Interface](#)

## Installing and Accessing the Script Queue Monitor

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Script Queue Monitor (Beta) is available as a SuiteApp which can be installed in your account. Its bundle ID is 56125 and it can be installed from production account 3923787.

### To install the Script Queue Monitor (Beta) SuiteApp:

1. Go to Customization > SuiteBundler > Search and Install Bundles.
2. Enter the bundle ID, **56125**, in the **Keywords** box and click the Search button.
3. Click the bundle link in the **Name** column to proceed to the Bundle Details Page.
4. On the Bundle Details page, click the **Install** button.
  - The Script Queue Monitor is a managed bundle. After you install it, future updates are automatically pushed to your account.
  - To proceed to the install, click OK in the popup that displays to obtain your consent for these future updates.
5. On the Preview Bundle Install Page, click the **Install Bundle** button.

After you have started the process of installing a bundle, the Installed Bundles page displays. If the installation is not complete, the **Status** column displays the percentage of installation progress. If the installation is complete, the **Status** column displays a green check.

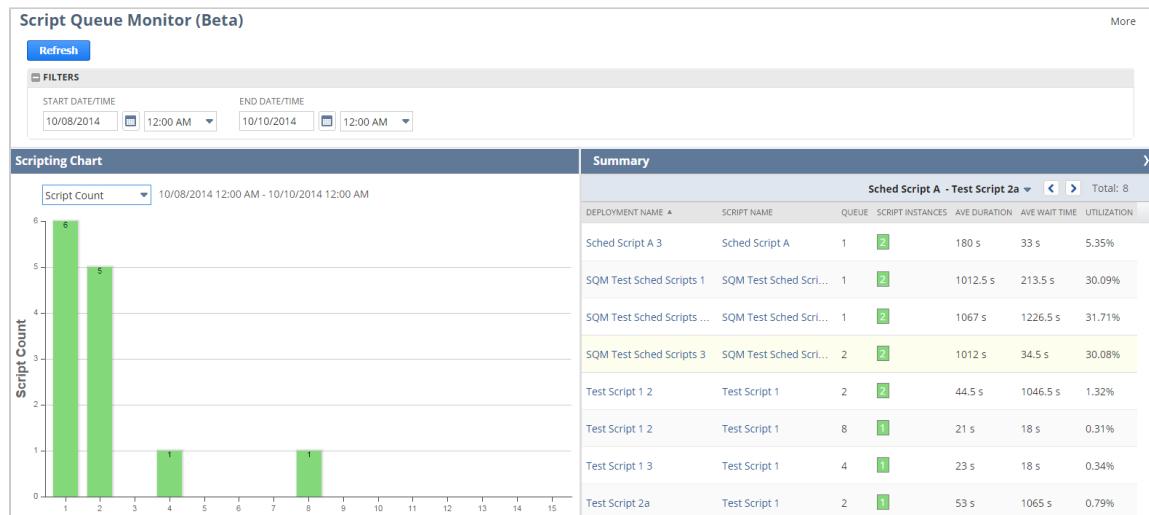
After this SuiteApp has been installed, the Script Queue Monitor is available at Customization > Script Performance > Script Queue Monitor, to account administrators only.

## Script Queue Monitor User Interface

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The UI for the Script Queue Monitor includes two panes: on the left, a pane that illustrates script instances for each queue, and on the right, a pane that summarizes relevant information about all script instances, aggregated by deployment name, script name, and queue.

In a Filters section at the top, you can select the time period for scripts to be included in the chart and list. Note that you can collapse and expand the Filters section as needed by clicking on its header bar. In the Scripting Chart pane, you can select the type of chart to display. You click the Refresh button to implement these selections in the Script Queue Monitor display.



## Summary Status Information

The grid in the right pane of the Script Queue Monitor includes the following details, aggregated by the combination of deployment name, script name, and queue number:

- Instances of that script deployment, on that queue, for the selected time period  
The color of a value in the Script Instances column indicates status, as shown in a key available from a hover over this value.
- Average duration, in seconds, of script execution time
- Average wait time, in seconds, before script execution
- Queue utilization percentage — the percentage of the total duration for all scripts running on that particular date and time

Summary						
Sched Script A - Test Script 2a ▾ < > Total: 8						
DEPLOYMENT NAME	SCRIPT NAME	QUEUE	SCRIPT INSTANCES	AVE DURATION	AVE WAIT TIME	UTILIZATION
Sched Script A 3	Sched Script A	1	2	180 s	33 s	5.35%
SQM Test Sched Scripts 1	SQM Test Sched Scri...	1	2	1012.5 s	213.5 s	30.09%
SQM Test Sched Scripts ...	SQM Test Sched Scri...	1	2	7 s	1226.5 s	31.71%
SQM Test Sched Scripts 3	SQM Test Sched Scri...	2	2	2 s	34.5 s	30.08%
Test Script 1 2	Test Script 1	2	2	5 s	1046.5 s	1.32%
Test Script 1 2	Test Script 1	8	1	21 s	18 s	0.31%
Test Script 1 3	Test Script 1	4	1	23 s	18 s	0.34%
Test Script 2a	Test Script 1	2	1	53 s	1065 s	0.79%

You can click on a column header to sort the summary by that column's values. Click two times to switch the sort from ascending to descending. Summary results are paginated for usability. Use the < and > buttons, or the dropdown arrow, to shift to a different page.

Summary						
Sched Script A - Test Script 2a ▾ < > Total: 8						

The Summary grid includes links you can click to drill down into more details:

- Click a link in the Deployment Name column to see the script deployment record.
- Click a link in the Script Name column to see the script record.
- Click on a value in the Script Instances column to display a popup window listing details about each script instance:

Scheduled Script Instance						
QUEUE	DEPLOYMENT NAME	SCRIPT NAME	10/8/2014 11:19 - 10/9/2014 12:03 ▾ < > Total: 2			
DATE CREATED	START	END	STATUS	% COMPLETE	DURATION	WAIT TIME
10/8/2014 11:19 ...	10/8/2014 11:23 ...	10/8/2014 11:39 ...	Compl...	100%	1010 s	191 s
10/9/2014 12:03 ...	10/9/2014 12:07 ...	10/9/2014 12:23 ...	Compl...	100%	1015 s	236 s

Click a link in the Deployment Name column to see the script deployment record.

Click a link in the Script Name column to see the script record.

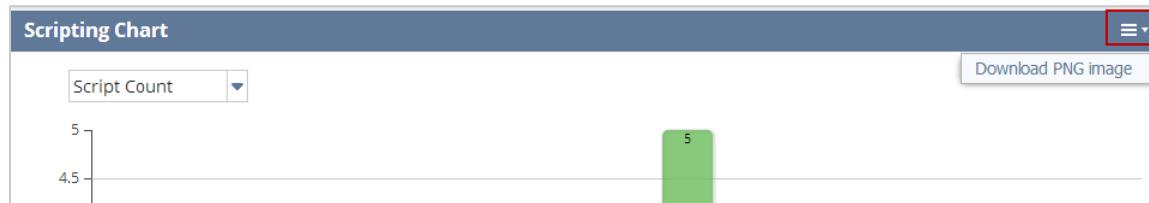
Click on a value in the Script Instances column to display a popup window listing details about each script instance:

## Scripting Chart

The Script Queue Monitor supports three types of charts:

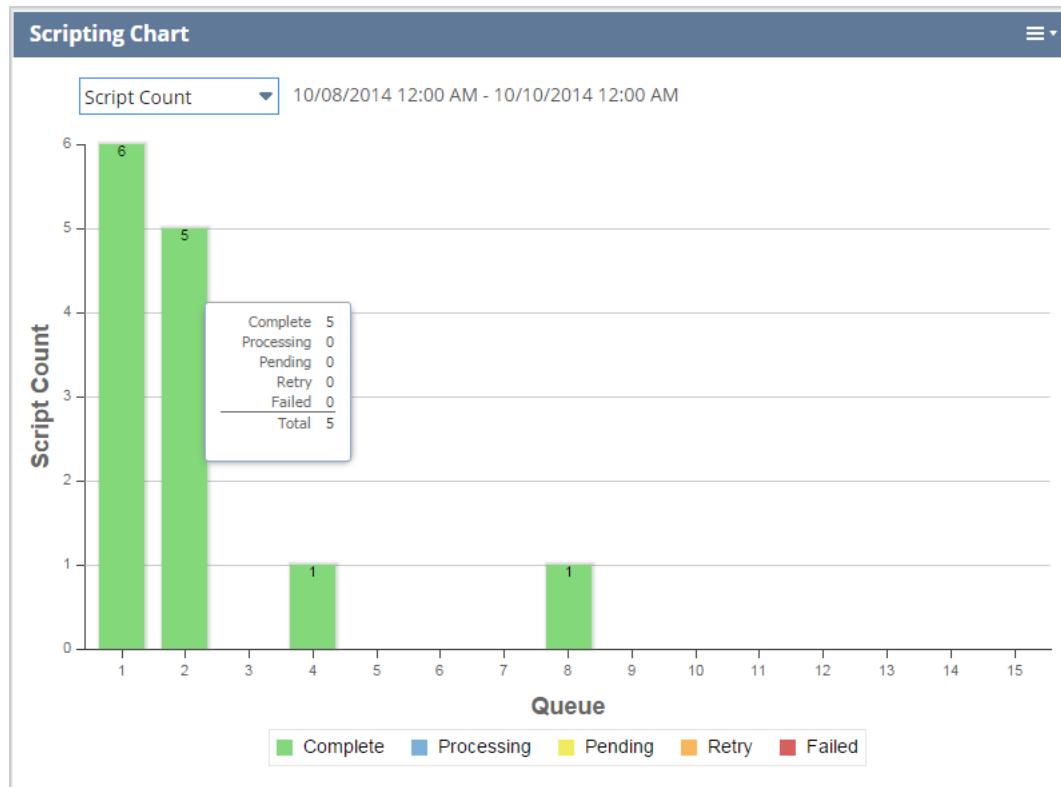
- [Script Count Chart](#)
- [Utilization Chart](#)
- [Timeline Chart](#)

Note that you can click the button available in the chart pane to download the currently displayed chart as a file named chart(<#>.png. (The # variable starts at 1 and is increased each time a chart is downloaded to avoid overwriting previously downloaded charts.)



## Script Count Chart

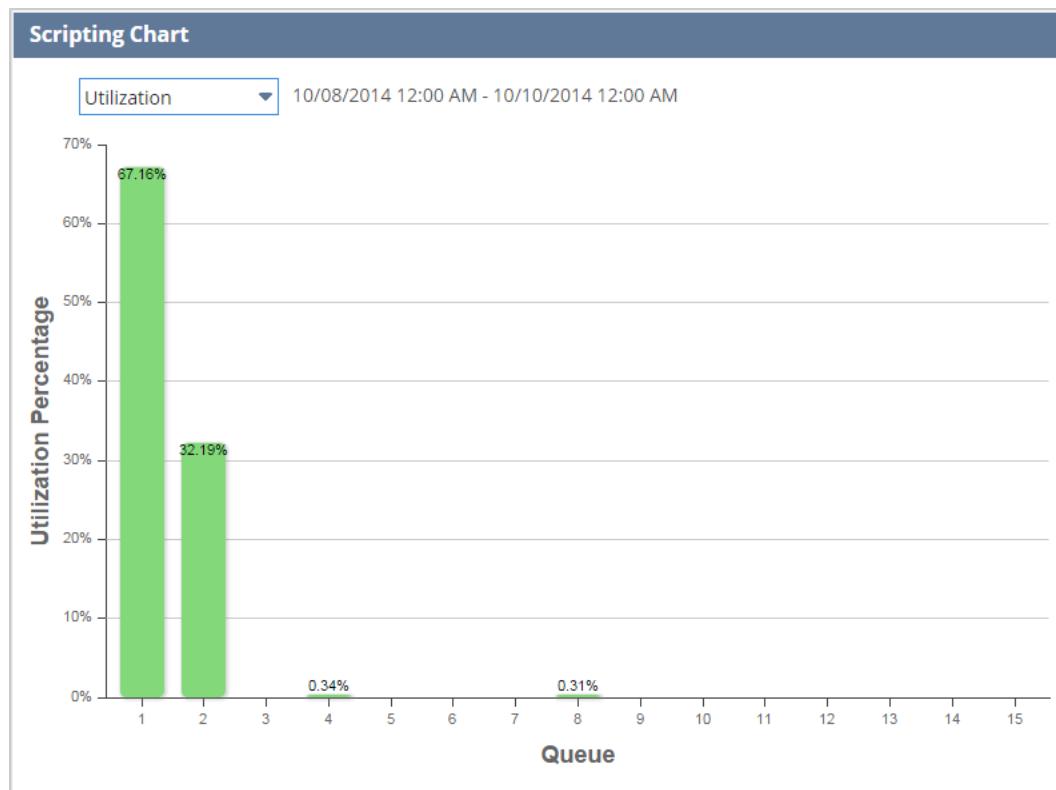
The Script Count chart displays vertical bars representing the number of scripts running on each queue during the selected time period. A hover over the bar for a queue displays a popup listing the number of scripts for each script status.



For each chart, bar color indicates script status, as explained by a legend at the bottom of the pane. You can click on a color in the legend to remove scripts with that status from the chart, and click on the color again to add scripts of that status back in.

## Utilization Chart

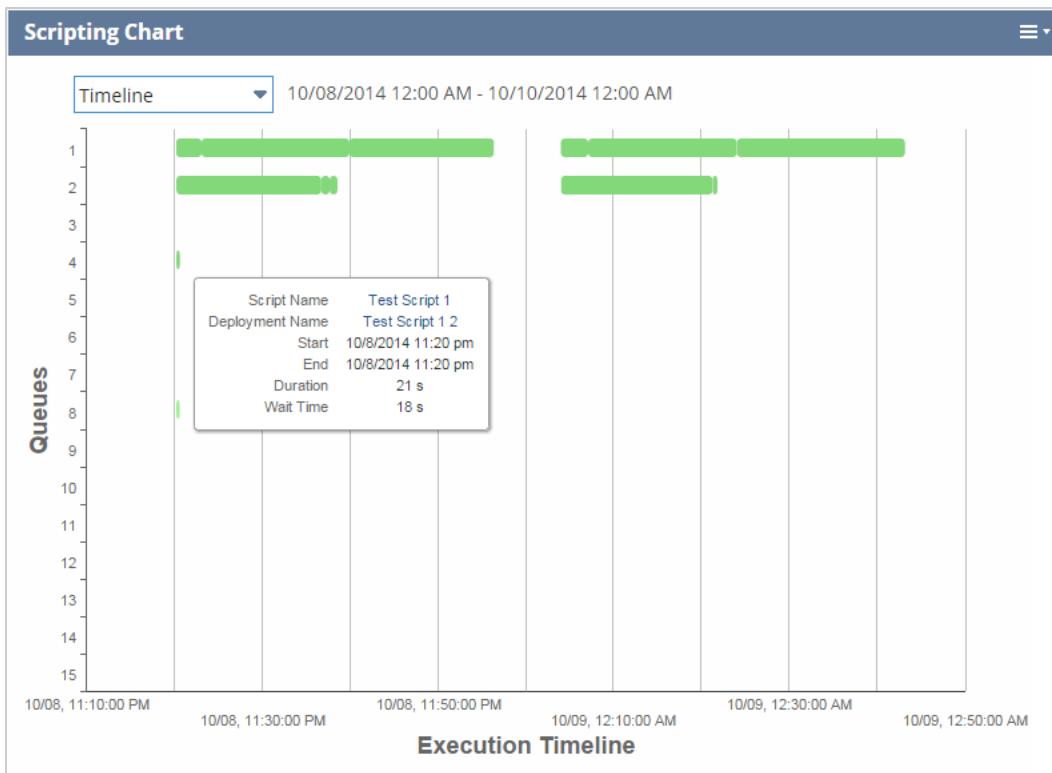
The Utilization chart displays vertical bars representing the percentage of capacity used for each queue during the selected time period. A hover over the bar for a queue displays a popup listing this percentage, the count of scripts run in the queue, and the total runtime of these scripts, in seconds.



## Timeline Chart

The Timeline chart displays horizontal bars representing the times during which scheduled scripts were running on each queue during the selected time period. A hover over the bar for a script displays a popup listing this script name, deployment name, start and end times, and total runtime, in seconds.

The Timeline chart displays horizontal bars representing the times during which scheduled scripts were running on each queue during the selected time period. A hover over the bar for a script displays a popup listing this script name, deployment name, start and end times, and total runtime in seconds.



You can grab and highlight a vertical slice of the timeline to zoom in and get a better view of script executions during the highlighted time period. To return to the higher level view, click the **Reset Zoom** button that is available in the lower right corner of the chart after a zoom.

**⚠️ Important:** The timeline chart cannot display a time period longer than 3 days. If the selected date range is longer, the timeline chart includes only the first 3 days.

# Working with Subrecords in SuiteScript

The following topics are covered in this section:

- What is a Subrecord?
- Using the SuiteScript API with Subrecords
- Creating and Accessing Subrecords from a Body Field
- Creating and Accessing Subrecords from a Sublist Field
- Setting Values on Subrecord Sublists
- Saving Subrecords Using SuiteScript
- Guidelines for Working with Subrecords in SuiteScript
- Working with Specific Subrecords in SuiteScript



**Note:** For a list of SuiteScript supported records and subrecords, see [SuiteScript Supported Records](#) in the NetSuite Help Center.

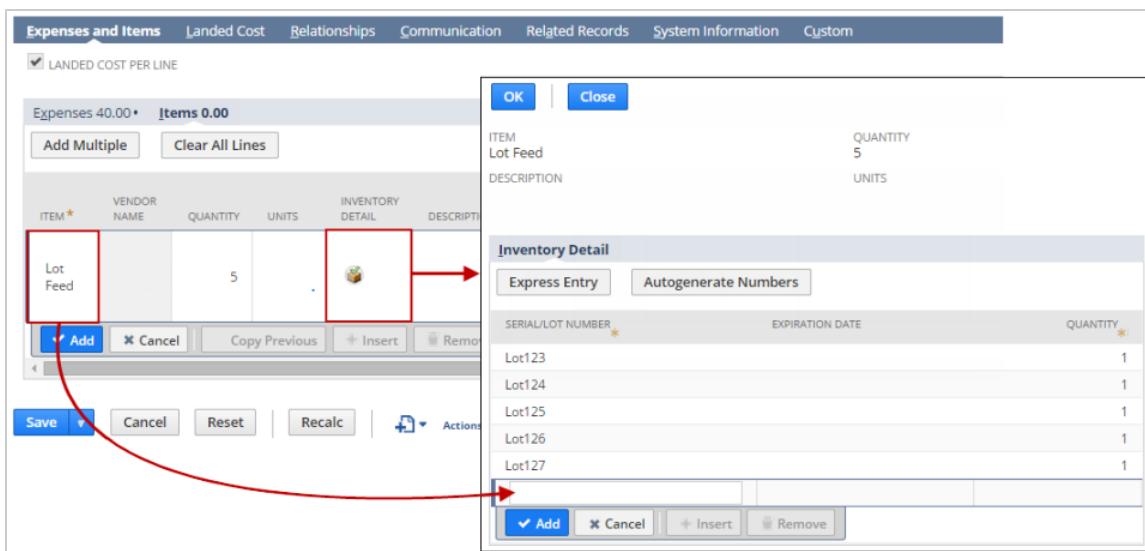
## What is a Subrecord?

A subrecord includes many of the same elements of a standard NetSuite record (body fields, sublists and sublist fields, and so on). However, subrecords must be created, edited, removed, or viewed from within the context of a standard (parent) record.

The purpose of a subrecord is to hold key related data about the parent record. For example, a parent record would be a Serialized Inventory Item record. This record defines a type of item. A subrecord would be an Inventory Detail subrecord. This is a subrecord that contains all data related to where the item might be stored in a warehouse. In this way, the subrecord contains data related to the item, but not data that directly defines the item. Without the parent record, the subrecord would serve no purpose.

The following figure shows an Inventory Detail subrecord. Its parent is a Bill record. In this figure the Inventory Detail subrecord is accessed through the Inventory Details sublist field. The Inventory Detail subrecord contains the inventory details for the item called the Lot Feed item.

In this case the parent record is still the Bill record, even though the subrecord tracks inventory details related to the Lot Feed item. Ultimately it is the Bill record that must be saved before the subrecord (pertaining to an item on the Bill) is committed to the database.



## Creating Subrecord Custom Entry Forms

You can create custom entry forms for subrecords by going to Setup > Customization > Entry Forms. A currently supported subrecord type is Inventory Detail, which is associated with the Advanced Bin / Numbered Inventory Management feature. In the Custom Entry Forms list, you can select Customize next to Inventory Detail to create a custom form for this subrecord type.

Note that when you create a custom form for Inventory Detail, you can use the Actions tab to add new buttons to the custom form. When clicked, these buttons will execute client SuiteScript. However, you cannot customize the buttons that currently exist on the Inventory Detail record. These buttons are required; without them you cannot save this subrecord to its parent record.

Also note that the Store Form with Record preference is not currently supported for custom subrecord forms. You can, however, set the customized subrecord form as Preferred.

Additionally, like any other custom form, you can attach client scripts to the Custom Forms tab.

## Using the SuiteScript API with Subrecords

The SuiteScript API includes several [Subrecord APIs](#) to interact with the subrecord object (`nlobjSubrecord`).

**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.

Using SuiteScript you can create and access subrecords through a **body field** on a parent record. (See [Creating and Accessing Subrecords from a Body Field](#) for details.) You can also create and access subrecords through a **sublist field** on a parent record. (See [Creating and Accessing Subrecords from a Sublist Field](#) for details.)

To set values on sublists that appear on subrecords, you will use some of the same Sublist APIs used to set values on sublists appearing on parent records. See [Setting Values on Subrecord Sublists](#) for details.

To save a subrecord, you must follow the pattern outlined in the section [Saving Subrecords Using SuiteScript](#).

# Creating and Accessing Subrecords from a Body Field

If you want to create a subrecord to hold data related to the parent, you can do so from a **body field** on the parent. When working with subrecords from a body field on the parent, you will use the following APIs if you are working with the parent record in a “current record” context, such as in a user event script or a client script:

- nlapiCreateSubrecord(fldname)
- nlapiEditSubrecord(fldname)
- nlapiRemoveSubrecord(fldname)
- nlapiViewSubrecord(fldname)

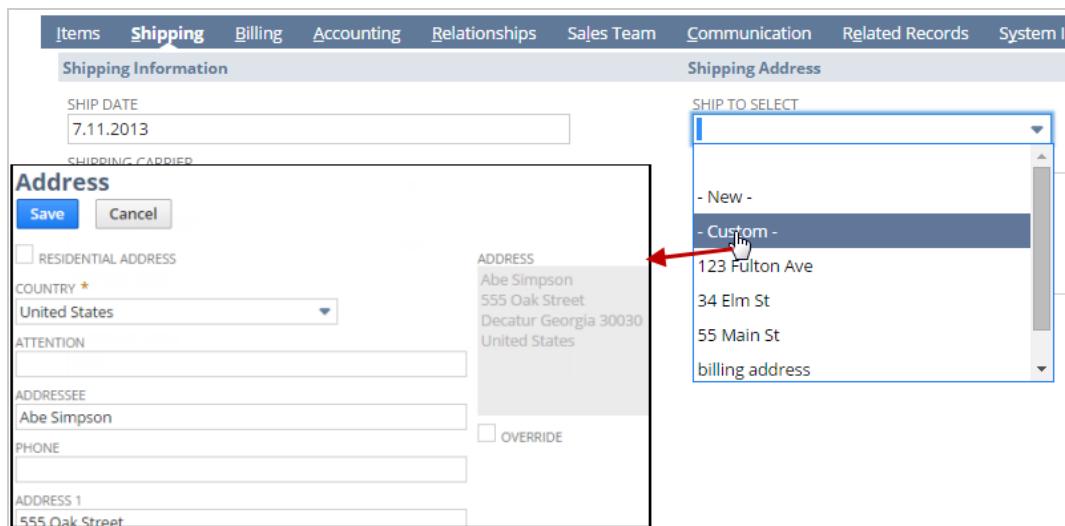
**Note:** nlapiCreateSubrecord(fldname) and nlapiEditSubrecord(fldname) are not supported in client scripts deployed on the parent record.

If you are loading the parent record using SuiteScript, you will use these methods on the `nlobjRecord` object to create and access a subrecord:

- createSubrecord(fldname)
- editSubrecord(fldname)
- removeSubrecord(fldname)
- viewSubrecord(fldname)

The following figure shows the Ship To Select (shippingaddress) body field on the Sales Order parent record. To create a custom shipping address subrecord on the parent, you will do so from this body field. After creating the subrecord, you can then edit, remove, or view the subrecord through the same body field on the parent record.

**Note:** For additional information on creating custom shipping addresses, see [Scripting Custom Billing and Shipping Addresses](#).



Note that after creating or editing a subrecord, you must save both the subrecord and the parent record for the changes to be committed to the database. See [Saving Subrecords Using SuiteScript](#) for more information.

For code samples showing how “body field” subrecord APIs are used, see [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#) or [Scripting Custom Billing and Shipping Addresses](#).

## Creating and Accessing Subrecords from a Sublist Field

If you want to create a subrecord to hold data for a record in a sublist, you can do so from a sublist field.

When working with subrecords from a **sublist field** on the parent record, you will use these APIs if you are working with the parent record in a “current record” context, such as in a user event script or a client script:

- `nlapiCreateCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiEditCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiViewCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiViewLineItemSubrecord(sublist, fldname, linenum)`

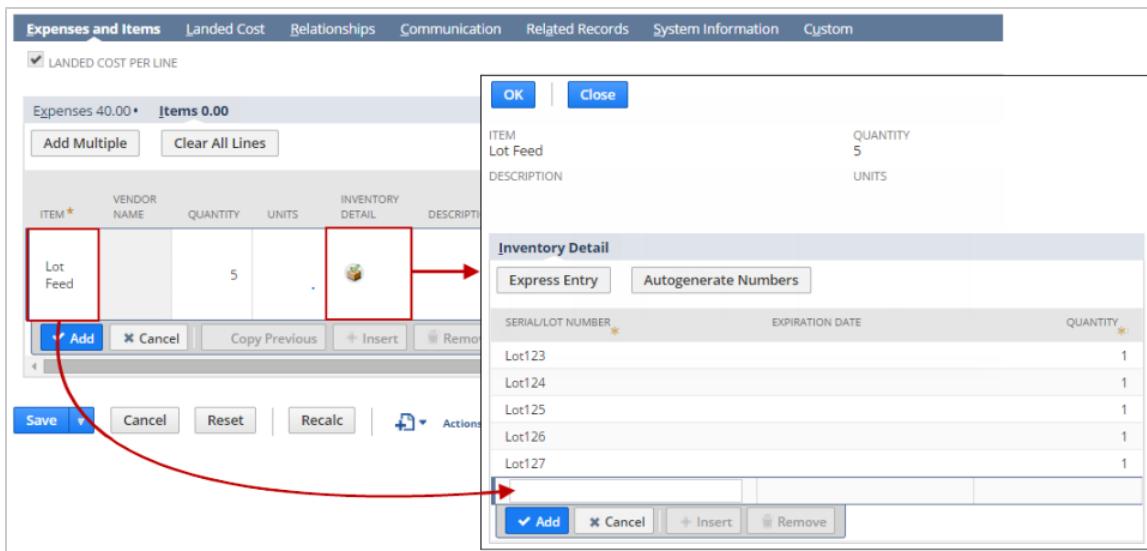


**Note:** `nlapiCreateCurrentLineItemSubrecord()` and `nlapiEditCurrentLineItemSubrecord()` are not currently supported in client scripts.

If you are loading the parent record using SuiteScript, and you want to create/access a subrecord from a sublist, you will use these methods on the `nlobjRecord` object:

- `createCurrentLineItemSubrecord(sublist, fldname)`
- `editCurrentLineItemSubrecord(sublist, fldname)`
- `removeCurrentLineItemSubrecord(sublist, fldname)`
- `viewCurrentLineItemSubrecord(sublist, fldname)`
- `viewLineItemSubrecord(sublist, fldname, linenum)`

This figure shows that the Inventory Detail subrecord is being edited on the Items sublist.

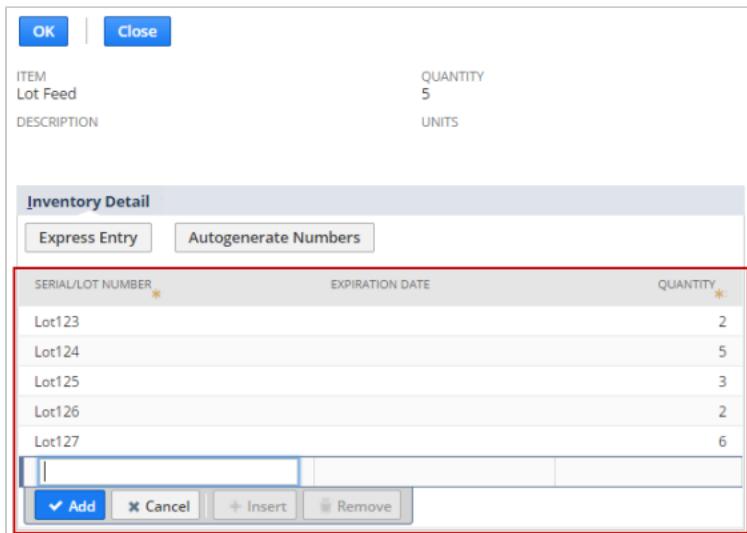


For code samples showing how “sublist field” subrecord APIs are used, see [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#).

## Setting Values on Subrecord Sublists

When working with sublists on subrecords (see figure), you will use the following Sublist APIs on the `nlobjRecord` object:

- `selectNewLineItem(group)` - use if creating a new sublist line
- `selectLineItem(group, linenumber)` - use if selecting an existing line on the sublist
- `setCurrentLineItemValue(group, name, value)` - use to set the values on a line
- `commitLineItem(group, ignoreRecalc)` - use to commit the line





**Important:** The nlapiSetLineItemValue(...) and nlobjRecord.setLineItemValue(...) APIs are NOT supported when scripting a subrecord's sublist.

The following sample shows how to use Sublist APIs to set values on a subrecord sublist.

```

var qtytobuild = 2;
var obj = nlapiCreateRecord('assemblybuild', {recordmode:'dynamic'});
obj.setFieldValue('subsidiary', 3 );
obj.setFieldValue('item', 174);
obj.setFieldValue('quantity', qtytobuild);
obj.setFieldValue('location', 2);

var bodySubRecord = obj.createSubrecord('inventorydetail');
var ctr;
for(ctr = 1; ctr <= qtytobuild ; ctr++)
{
    //Here we are selecting a new line on the Inventory Assignment sublist on the subrecord
    bodySubRecord.selectNewLineItem('inventoryassignment');
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'newinventorynumber',
        'amsh_' + ctr);
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 1);
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'binnumber', 3);
    bodySubRecord.commitLineItem('inventoryassignment');
}
bodySubRecord.commit();

//Here we are selecting and editing an existing line on the Components sublist
//on the parent record. Note that when working with the Assembly Build record only,
//the internal ID for the Inventory Details field on the Components sublist is
// 'componentinventorydetail'. This is because the Assembly Build record already contains
//an Inventory Details (inventorydetails) body field.
obj.selectLineItem('component', 1);
obj.setCurrentLineItemValue('component', 'quantity', qtytobuild);
var compSubRecord = obj.createCurrentLineItemSubrecord('component',
    'componentinventorydetail');

//Here we are selecting and editing a new line on the Inventory Assignment sublist on
//the subrecord.
compSubRecord.selectNewLineItem('inventoryassignment');
compSubRecord.setCurrentLineItemValue('inventoryassignment', 'binnumber', 3);
compSubRecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 2);
compSubRecord.commitLineItem('inventoryassignment');
compSubRecord.commit();

obj.commitLineItem('component');
var id = nlapiSubmitRecord(obj);
obj = nlapiLoadRecord('assemblybuild', id);
var subrecord = obj.viewSubrecord('inventorydetail');
subrecord.selectLineItem('inventoryassignment', 1);

var str;
```

```

str = subrecord.getCurrentLineItemValue('inventoryassignment', 'newinventorynumber');
if (str!= 2)
{
}

```

For additional code samples showing how to use Sublist APIs in the context of a subrecord, see [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#).

## Saving Subrecords Using SuiteScript

To save a subrecord to a parent record you will call `nlobjSubrecord.commit()`. 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.

In the following sample an Inventory Detail subrecord is edited from the 'inventorydetail' field on the Items sublist. Next, values are set on the 'inventoryassignment' sublist. This is the sublist on the Inventory Detail subrecord. After this sublist is edited, you must call `commitLineItem(...)` to commit the changes to this sublist.

Next, you call `commit()` on the `nlobjSubrecord` object to commit the subrecord to the parent record. After that, you must call `commitLineItem(...)` again, but this time on the Items sublist of the parent record. This is necessary because, ultimately what you are doing in this script is updating the Items sublist.

Finally, you must call `nlapiSubmitRecord(...)` on the Purchase Order record. This is the parent record and must be saved for all changes in the script to be committed to the database.

```

var record2= nlapiLoadRecord('purchaseorder', id, {recordmode: 'dynamic'});
record2.selectLineItem('item', 1);
record2.setCurrentLineItemValue('item', 'quantity', 2);

var subrecord2= record2.editCurrentLineItemSubrecord("item", "inventorydetail");
subrecord2.selectLineItem('inventoryassignment', 1);
subrecord2.setCurrentLineItemValue('inventoryassignment', 'inventorynumber', 'working123');
subrecord2.selectNewLineItem('inventoryassignment');
subrecord2.setCurrentLineItemValue('inventoryassignment', 'inventorynumber', '2ndlineinventorynumber');
subrecord2.setCurrentLineItemValue('inventoryassignment', 'quantity', '1');
subrecord2.commitLineItem('inventoryassignment');

subrecord2.commit();

record2.commitLineItem('item');

var id = nlapiSubmitRecord(record2);

```

## Guidelines for Working with Subrecords in SuiteScript

The following are guidelines you must follow when working with subrecords.

- In SuiteScript, you must first create or load a parent record before you can create/access a subrecord. You can create/load the parent record in either standard mode or dynamic mode.
- You cannot create or edit a subrecord in a beforeLoad user event script. You must use a pageInit client script if you want to create/edit a subrecord before the end user has access to the page.
- If you attempt to edit or view a subrecord that does not exist, `null` will be returned.
- In a client script attached or deployed to the parent record, you cannot create or edit a subrecord; you can only view or delete subrecords.
- There is no automatic client-side validation on a subrecord when a field is changed on the parent record. For example, if a user changes the quantity of an item on an item line, there is no detection of a quantity mismatch between the item line and its Inventory Detail. Note, however, validation can be implemented programmatically using a `validateLine()` call.
- To save a subrecord, you must commit both the subrecord, the line the subrecords appears on (if accessing a subrecord through a sublist), and the parent record. See [Saving Subrecords Using SuiteScript](#) for complete details.
- If you call one of the [Subrecord APIs](#) on a non-subrecord field, an error is thrown.
- The following sublist and body field APIs are not supported on subrecords:
  - `nlapiGetLineItemValue(type, fldname, linenum)`
  - `nlapiGetLineItemText(type, fldnam, linenum)`
  - `nlapiFindLineItemValue(type, fldnam, val)`
  - `nlapiGetCurrentLineItemText(type, fldnam)`
  - `nlapiGetCurrentLineItemValue(type, fldnam)`
  - `nlapiGetFieldValue()`
  - `nlapiGetFieldText()`
- When using the Assembly Build record as a parent record, be aware that this record has two `inventorydetail` fields: one on the body of the record and the other as a field on the Components sublist. When creating/assessing a subrecord from the body field, use `inventorydetail` as the internal ID for the `fldname` parameter. When creating/accessing a subrecord from the sublist field on the Components sublist, use `componentinventorydetail` as the internal ID for the `fldname` parameter. To see an example, see the code sample provided in [Setting Values on Subrecord Sublists](#).

## Working with Specific Subrecords in SuiteScript

- Using SuiteScript with Advanced Bin / Numbered Inventory Management
- Using SuiteScript with Address Subrecords

## Using SuiteScript with Advanced Bin / Numbered Inventory Management

When you write scripts with the Advanced Bin / Numbered Inventory Management feature enabled, your scripts must reference not only a main (parent) record or transaction, but also the Inventory Details "subrecord." In the UI the Inventory Details subrecord appears as a pop-up when you click the Inventory Details body field or sublist field. In SuiteScript, this pop-up is considered a subrecord object (`nlobjSubrecord`), which is created and accessed through its own set of [Subrecord APIs](#).

See the following topics for details specific to using SuiteScript with the Advanced Bin / Numbered Inventory Management feature:

- SuiteScript and Advanced Bin Management – Overview
- Scripting the Inventory Detail Subrecord
- Sample Scripts for Advanced Bin / Numbered Inventory Management

See these topics for general information on working with subrecords:

- Working with Subrecords in SuiteScript
- Guidelines for Working with Subrecords in SuiteScript

**✖ Warning:** If you are currently using SuiteScript with the **basic** Bin Management feature, your scripts will no longer work after you enable the Advanced Bin / Numbered Inventory Management feature. This is especially true if you have written client scripts, which will have to be completely rewritten as server scripts. See [Updating Your Scripts After Enabling Advanced Bin / Numbered Inventory Management](#) for more information.

## SuiteScript and Advanced Bin Management – Overview

The following figure draws a comparison between how the advanced bin management feature appears in the UI and how that translates into SuiteScript.

**i Note:** Even with the Advanced Bin / Numbered Inventory Management feature enabled, not all items will require an Inventory Details subrecord. (See the help topic [Advanced Bin / Numbered Inventory Management](#) for details on which items will use Inventory Detail subrecords.)

The figure below shows a subrecord being accessed from a sublist field. (Note that subrecords can also be created and accessed from a body field on the parent record. See [Creating and Accessing Subrecords from a Body Field](#) for more information.)

The numbers below further explain the numbers in the figure.

1. **Bill** : This is a parent record. In both the UI and in SuiteScript you must have a parent record before you can create or access a subrecord. Without the parent, the subrecord has no relevance.  
In SuiteScript, you can create/load the parent record in either standard mode or dynamic mode.
2. **Items sublist** : In the case of this figure, a subrecord is being created for the Lot Bin Item referenced on the Items sublist.

**⚠ Important:** Note that the parent is still considered to be the Bill record, even though the subrecord is being created for the Lot Bin Item. Ultimately it is the Bill record that must be saved before any changes to the Items sublist or the Inventory Details subrecord are committed to the database.

3. **Inventory Details sublist field** : As you enter information for the Lot Bin Item, in the UI you click the Inventory Details icon to create a new Inventory Details subrecord for this item.  
In SuiteScript, if you are creating a subrecord from the Inventory Details sublist field, you will call either `nlapicreateCurrentLineItemSubrecord(sublist, fldname)` or `nlobjRecord.createCurrentLineItemSubrecord(sublist, fldname)`, depending on the nature of your script. The sublist is 'item' and the fldname is 'inventorydetail'.
4. **Inventory Detail subrecord** : This is a subrecord containing the inventory details for the Lot Bin Item.
5. **Inventory Assignment sublist** : This is the sublist that appears on the Inventory Details subrecord. Although there is no UI label for the Inventory Assignment sublist, this is the sublist you will reference to add and edit new sublist lines.

In SuiteScript, you create and edit lines on the Inventory Assignment sublist using the APIs described in [Setting Values on Subrecord Sublists](#).

6. **OK button:** In the UI you click the OK button to save the subrecord (note, however, the subrecord is not yet saved on the server).

In SuiteScript, to save a subrecord you must call `nlobjSubrecord.commit()` on the subrecord, `nlobjRecord.commitLineItem()` - - if you have created a subrecord on a sublist line.

7. **Add button on sublist:** In the UI you must click the Add button on a sublist to commit your changes to the line.

In SuiteScript, you will call `nlobjRecord.commitLineItem()` - - if you have created a subrecord on a sublist line.

8. **Save button on parent record:** In the UI, you will click the Save button on the parent record to commit all changes to the server.

In SuiteScript, you will call `nlaplSubmitRecord(...)` on the parent record. See [Saving Subrecords Using SuiteScript](#) for complete details.

## Scripting the Inventory Detail Subrecord

Like a standard (parent) record, the Inventory Details subrecord contains body fields, a sublist, and sublist fields. You can create and access the Inventory Details subrecord from a body field on the

parent record or from a sublist field. See [Creating and Accessing Subrecords from a Body Field](#) and [Creating and Accessing Subrecords from a Sublist Field](#) for details.

**Note:** Currently you cannot create or edit subrecords using Client SuiteScript.

The sublist that appears on the Inventory Detail subrecord is referred to as the Inventory Assignment sublist, even though it has no UI label. In your scripts, use the ID `inventoryassignment` to reference this sublist. In the figure below, the `inventoryassignment` sublist is used to assign serial/lot numbers and which serial/lot belongs to which bin.

The screenshot shows the 'Inventory Detail' subrecord in a SuiteScript interface. At the top, there are fields for 'ITEM' (Lot Feed) and 'QUANTITY' (5). Below this is a table titled 'Inventory Detail' with two tabs: 'Express Entry' and 'Autogenerate Numbers'. The table has columns for 'SERIAL/LOT NUMBER', 'EXPIRATION DATE', and 'QUANTITY'. A red box highlights the data rows:

SERIAL/LOT NUMBER *	EXPIRATION DATE	QUANTITY *
Lot123		2
Lot124		5
Lot125		3
Lot126		2
Lot127		6

At the bottom of the sublist table are buttons: 'Add', 'Cancel', 'Insert', and 'Remove'. The entire sublist area is enclosed in a red border.

To set values on the Inventory Assignment sublist, you will use some of the same Sublist APIs used to set values on other sublist in the system. See [Setting Values on Subrecord Sublists](#) for details.

To save a subrecord, you must follow the pattern outlined in the section [Saving Subrecords Using SuiteScript](#).

## Internal IDs for the Inventory Details Subrecord

Use the following internal IDs when writing SuiteScript against the Inventory Details subrecord:

Subrecord Elements	Internal IDs	Notes
Inventory Details subrecord	<code>inventorydetail</code>	<p>This is the internal ID for the Inventory Details subrecord.</p> <p>When using any of the <a href="#">Subrecord APIs</a>, the value you set for the <code>fldname</code> parameter is <code>inventorydetail</code>. It is through this field that you will create a subrecord.</p>
<p><b>Important:</b> When using the Assembly Build record as a parent record, be aware that this record has two <code>inventorydetail</code> fields: one on the body of the record and the other as a field on the Components sublist.</p> <p>When creating/assessing a subrecord from the body field, use <code>inventorydetail</code> as the internal ID for the <code>fldname</code> parameter.</p>		

Subrecord Elements	Internal IDs	Notes
		When creating/accessing a subrecord from the sublist field on the Components sublist, use <b>componentinventorydetail</b> as the internal ID for the <b>fldname</b> parameter. To see an example, see the code sample provided in <a href="#">Setting Values on Subrecord Sublists</a> .
Body fields on the Inventory Details subrecord	item location tolocation itemdescription quantity baseunitquantity unit totalquantity	
Inventory Assignment sublist	inventoryassignment	This is the internal ID for the Inventory Assignment sublist that appears on the Inventory Details subrecord. Note that the Inventory Assignment has no UI label.
Sublist fields on the Inventory Assignment sublist	receiptinventorynumber issueinventorynumber binnumber tobinnumber expirationdate quantity quantityavailable	This field is for entering text of serial/lot number for create.  This is the select field where users pick an issueinventorynumber out of inventory

## Sample Scripts for Advanced Bin / Numbered Inventory Management

The following samples are provided in this section:

- [Creating an Inventory Detail Subrecord](#)
- [Editing an Inventory Detail Subrecord](#)
- [Removing an Inventory Detail Subrecord from a Sublist Line](#)
- [Canceling an Inventory Detail Subrecord](#)
- [Viewing an Inventory Detail Subrecord](#)
- [Updating Your Scripts After Enabling Advanced Bin / Numbered Inventory Management](#)

As you begin writing your own scripts, NetSuite strongly recommends that you see [Guidelines for Working with Subrecords in SuiteScript](#). This section highlights many of the rules that are enforced when scripting with subrecords.

## Creating an Inventory Detail Subrecord

This sample shows how to create a subrecord from a body field and from a sublist field. The subrecord created on the body field is an Inventory Detail subrecord pertaining to the Assembly Build (parent) record.

The subrecord created on the sublist field (using `nlobjRecord.createCurrentLineItemSubrecord(sublist, fldname)`) is an Inventory Detail subrecord. This subrecord pertains to the first component on the Components sublist (of the Assembly Build parent record).

Notice that to set values on the sublist for the Inventory Detail (the Inventory Assignment sublist), you will use many of the same APIs you use to work with sublists on a parent record.

```
var qtytobuild = 2;
var obj = nlapiCreateRecord('assemblybuild', {recordmode:'dynamic'});
obj.setFieldValue('subsidiary', 3 );
obj.setFieldValue('item', 174);
obj.setFieldValue('quantity', qtytobuild);
obj.setFieldValue('location', 2);

var bodySubRecord = obj.createSubrecord('inventorydetail');
var ctr;
for(ctr = 1; ctr <= qtytobuild ; ctr++)
{
    bodySubRecord.selectNewLineItem('inventoryassignment');
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'receiptinventorynumber',
    'amsh_' + ctr);
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 1);
    bodySubRecord.setCurrentLineItemValue('inventoryassignment', 'binnumber', 3);
    bodySubRecord.commitLineItem('inventoryassignment');
}
bodySubRecord.commit();

obj.selectLineItem('component', 1);
obj.setCurrentLineItemValue('component', 'quantity', qtytobuild);
var compSubRecord = obj.createCurrentLineItemSubrecord('component',
'componentinventorydetail');

compSubRecord.selectNewLineItem('inventoryassignment');
compSubRecord.setCurrentLineItemValue('inventoryassignment', 'binnumber', 3);
compSubRecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 2);
compSubRecord.commitLineItem('inventoryassignment');
compSubRecord.commit();

obj.commitLineItem('component');
var id = nlapiSubmitRecord(obj);
```



**Important:** To save a subrecord you must call commit() on the subrecord, commitLineItem()-- if you have created a subrecord on a sublist line - - AND nlapiSubmitRecord(...) on the parent record. See [Saving Subrecords Using SuiteScript](#) for more details.

## Editing an Inventory Detail Subrecord

To edit a subrecord you must first load the parent record. In the sample below the parent record (a Purchase Order) is loaded in dynamic mode. When working with subrecords, you can load parent records in dynamic mode or in standard mode. However, the subrecord itself must be scripted using "dynamic" subrecord APIs. These are the APIs that have "current" in the function or method signature.



**Note:** If you attempt to edit or view a subrecord that does not exist, null is returned.

```
var record2= nlapiLoadRecord('purchaseorder', id, {recordmode: 'dynamic'});
record2.selectLineItem('item', 1);
record2.setCurrentLineItemValue('item', 'quantity', 2);

var subrecord2= record2.editCurrentLineItemSubrecord('item', 'inventorydetail');
subrecord2.selectLineItem('inventoryassignment', 1);
subrecord2.setCurrentLineItemValue('inventoryassignment', 'issueinventorynumber', 'working123')
;
subrecord2.selectNewLineItem('inventoryassignment');
subrecord2.setCurrentLineItemValue('inventoryassignment', 'issueinventorynumber',
    '2ndlineinventorynumber');
subrecord2.setCurrentLineItemValue('inventoryassignment', 'quantity', '1');
subrecord2.commitLineItem('inventoryassignment');

subrecord2.commit();

record2.commitLineItem('item');

var id = nlapiSubmitRecord(record2);
```

## Removing an Inventory Detail Subrecord from a Sublist Line

The following sample shows how to remove a subrecord from a sublist line with removeCurrentLineItemSubrecord.

```
var purchaseOrder = nlapiLoadRecord('purchaseorder', 1792, {recordmode: 'dynamic'});
var i=1;
var totalLine = purchaseOrder.getLineItemCount('item');

for(i; i<=totalLine; i++)
{
    purchaseOrder.selectLineItem('item', i);
    var invDetailSubrecord = purchaseOrder.viewCurrentLineItemSubrecord('item',
        'inventorydetail');
    if(invDetailSubrecord != null)
    {
        purchaseOrder.removeCurrentLineItemSubrecord('item', 'inventorydetail');
        purchaseOrder.commitLineItem('item');
```

```

    }
}

nlapiSubmitRecord(purchaseOrder);

```

Note that the `nlapiRemoveSubrecord(fldname)` and `nlobjRecord.removeSubrecord(fldname)` APIs are for removing subrecords from a body field on the parent record. Assembly Build and Assembly Unbuild are the only two parent record types that support the creation of a subrecord on a body field. Therefore, these APIs would only be useful in the context of these two record types. Be aware though that even in the UI, NetSuite business logic prevents users from removing subrecords from these parents when the subrecords are created from a body field. This means that in SuiteScript, if you attempt to call either of the `removeSubrecord` body field APIs, and then you call `nlapiSubmitRecord` on the parent, a user error will be thrown. This is in adherence to NetSuite business logic.

If you want to use either of the `removeSubrecord` body field APIs, it will probably be in the context of creating, and then removing your subrecord all in the same code, based on your particular use case.

## Cancelling an Inventory Detail Subrecord

The following sample shows how to cancel the submission of a subrecord.

```

var purchaseOrder=nlapiCreateRecord('purchaseorder', {recordmode: 'dynamic'});
purchaseOrder.setFieldValue('entity', 38);
purchaseOrder.selectNewLineItem('item');
purchaseOrder.setCurrentLineItemValue('item', 'item', 909 );
purchaseOrder.setCurrentLineItemValue('item', 'quantity', 1);

var invDetailSubrecord = purchaseOrder.createCurrentLineItemSubrecord('item', 'inventorydetail'
);
invDetailSubrecord.selectNewLineItem('inventoryassignment');
invDetailSubrecord.setCurrentLineItemValue('inventoryassignment', 'receiptinventorynumber', 'E1
OJNF98');

invDetailSubrecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 1);
invDetailSubrecord.commitLineItem('inventoryassignment');
invDetailSubrecord.cancel(); //undo this subrecord operation

purchaseOrder.commitLineItem("item"); // no subrecord is saved with this line.
var test = nlapiViewLineItemSubrecord('item', 'inventorydetail', 1);

nlapiLogExecution('DEBUG', 'subrecord should be null, and it is: ' +test);
nlapiSubmitRecord(purchaseOrder);

```

## Viewing an Inventory Detail Subrecord

The following samples show how to use different versions of the subrecord “view subrecord” APIs.

### Example 1

This sample shows how to return the read-only details of the subrecord that appears on the first line (which is the *current* line) of the Items sublist.

The sample also shows how get the read-only details of the subrecord associated with the second line on the sublist.

```

var purchaseOrder=nlapiLoadRecord('purchaseorder', 1793, {recordmode: 'dynamic'});
purchaseOrder.selectLineItem('item', 1);
var invDetailSubrecord = purchaseOrder.viewCurrentLineItemSubrecord('item', 'inventorydetail');

invDetailSubrecord.selectLineItem('inventoryassignment', 1);

nlapiLogExecution('DEBUG', 'inventory number: ' +
    invDetailSubrecord.getCurrentLineItemValue('inventoryassignment',
    'receiptinventorynumber'));

var invDetailOnLine2 = purchaseOrder.viewLineItemSubrecord('item', 'inventorydetail', 2);
invDetailOnLine2.selectLineItem('inventoryassignment', 1);

nlapiLogExecution('DEBUG', 'inventory number: ' +
    invDetailOnLine2.getCurrentLineItemValue('inventoryassignment',
    'receiptinventorynumber'));

```

## Example 2

This sample shows how to use the view API to access a subrecord associated with a body field.

```

var record3 = nlapiLoadRecord('assemblybuild', id, {recordmode: 'dynamic'});
var subrecord3 = record3.viewSubrecord('inventorydetail');
subrecord3.selectLineItem('inventoryassignment', 1);

nlapiLogExecution('DEBUG', 'inven: ' + subrecord3.getCurrentLineItemValue('inventoryassignment'
, 'issueinventorynumber'));

```

## Updating Your Scripts After Enabling Advanced Bin / Numbered Inventory Management

The first script shows what a typical script might look like with the Advanced Bin / Numbered Inventory Management feature turned off (not enabled). Notice that in this script you are calling the `setCurrentLineItemValue(...)` API to set inventory and serial number details for the item.

When scripting with the advanced bin management feature enabled, these lines of code will break. Instead, you must create subrecords to hold all inventory detail data.

### With Advanced Bin / Numbered Inventory Management OFF

```

var obj = nlapiCreateRecord('inventoryadjustment');
obj.setFieldValue('subsidiary', 3); //UK
obj.setFieldValue('account', 173 );
obj.setFieldValue('department', 2);
obj.setFieldValue('class', 2);
obj.setFieldValue('memo', 'Testing 123');
obj.setFieldValue('adjlocation' , 2);

obj.selectNewLineItem('inventory');
obj.setCurrentLineItemValue('inventory', 'item', 170);
obj.setCurrentLineItemValue('inventory', 'location', 2);
obj.setCurrentLineItemValue('inventory', 'adjustqtyby', 1);

```

```
//The next lines will be break when adv. bin management is turned on.  
obj.setCurrentLineItemValue('inventory', 'serialnumbers', 'testserial');  
obj.setCurrentLineItemValue('inventory', 'binnumbers', 'bin1');  
obj.commitLineItem('inventory');  
  
var id = nlapiSubmitRecord(obj);
```

## With Advanced Bin / Numbered Inventory Management ON

The following shows the changes you would have to make to your script to account for the new subrecord object model.

```
var obj = nlapiCreateRecord('inventoryadjustment', {recordmode:'dynamic'});  
  
obj.setFieldValue('subsidiary', 3); //UK  
obj.setFieldValue('account', 173);  
obj.setFieldValue('department', 2);  
obj.setFieldValue('class', 2);  
obj.setFieldValue('memo', 'Testing 123');  
obj.setFieldValue("adjlocation", 2);  
  
obj.selectNewLineItem('inventory');  
obj.setCurrentLineItemValue('inventory', 'item', 170);  
obj.setCurrentLineItemValue('inventory', 'location', 2);  
obj.setCurrentLineItemValue('inventory', 'adjustqtyby', 1);  
  
// the setCurrentLineItemValue API used in the first example must now be removed,  
// and a subrecord must be created to hold the data you want  
  
var subrecord = obj.createCurrentLineItemSubrecord('inventory', 'inventorydetail');  
  
subrecord.selectNewLineItem('inventoryassignment');  
subrecord.setCurrentLineItemValue('inventoryassignment', 'receiptinventorynumber',  
'testserial');  
subrecord.setCurrentLineItemValue('inventoryassignment', 'quantity', 1);  
subrecord.setCurrentLineItemValue('inventoryassignment', 'binnumber', 'bin1');  
subrecord.commitLineItem('inventoryassignment');  
subrecord.commit();  
  
obj.commitLineItem('inventory');  
  
var id = nlapiSubmitRecord(obj);
```

## Using SuiteScript with Address Subrecords

See the following topics for details specific to using SuiteScript with the Address Customization feature:

- SuiteScript and Address Subrecords – Overview
- Scripting the Address Subrecord
- Sample Scripts for Address Subrecords
- Scripting Custom Billing and Shipping Addresses

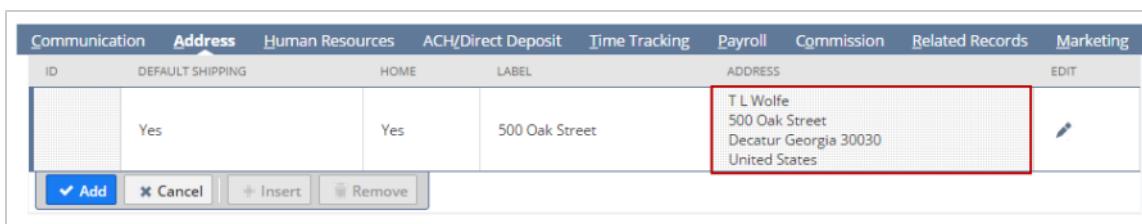
See these topics for general information on working with subrecords:

- Working with Subrecords in SuiteScript
- Guidelines for Working with Subrecords in SuiteScript

## SuiteScript and Address Subrecords – Overview

The Address Customization feature consolidates individual address fields into an address subrecord. Within SuiteBuilder, you can create custom address forms (templates) for different countries. These forms determine the fields available on the address subrecord (for example, a UK address has different fields than a US address). When creating a new address, you create a new sublist item for it. You then choose the address form you want to use by selecting the country associated with it. See the help topic [Working with Addresses on Transactions](#) for additional information.

The address subrecord is accessed from a sublist field on the parent record. In the following screenshot, the sublist field that contains the address subrecord is outlined in red.



Fields on the address sublist are not part of the address subrecord. To access the address subrecord fields, you must open the subrecord. Within the UI, you access the address subrecord by clicking the pencil icon in the Edit sublist field. The address subrecord is shown below.

The screenshot shows an 'Address' dialog box with the following fields:

- COUNTRY \***: United States
- ATTENTION**: (empty)
- ADDRESSEE**: T L Wolfe
- PHONE**: 555-555-5555
- ADDRESS 1**: 500 Oak Street
- ADDRESS 2**: (empty)
- CITY**: Decatur
- STATE**: Georgia
- ZIP**: 30030

On the right side of the dialog, there is a section labeled 'ADDRESS' containing the address details: T L Wolfe, 500 Oak Street, Decatur Georgia 30030, United States. Below this is a 'Map' button. There is also an 'OK' button at the top left and a 'Cancel' button at the top right.

## Scripting the Address Subrecord

Address subrecords are accessed from a sublist field on the parent record. When scripting address subrecords, use the following APIs if you are working with the parent record in a "current record" context, such as in a user event script or a client script:

- `nlapiCreateCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiEditCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiRemoveCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiViewCurrentLineItemSubrecord(sublist, fldname)`
- `nlapiViewLineItemSubrecord(sublist, fldname, linenum)`

**Note:** `nlapiCreateCurrentLineItemSubrecord` and `nlapiEditCurrentLineItemSubrecord` are not currently supported in client scripts.

If you are loading the parent record using SuiteScript, and you want to create/access a subrecord from a sublist, use these methods on the `nlobjRecord` object:

- `createCurrentLineItemSubrecord(sublist, fldname)`
- `editCurrentLineItemSubrecord(sublist, fldname)`
- `removeCurrentLineItemSubrecord(sublist, fldname)`
- `viewCurrentLineItemSubrecord(sublist, fldname)`
- `viewLineItemSubrecord(sublist, fldname, linenum)`

Access the sublist and subrecord with the following internal ID names:

- Sublist – 'addressbook'
- Address Subrecord – 'addressbookaddress'

There are two state fields available in addresses:

- `state` is a text entry field that is not validated
- `dropdownstate` is a select field that can be used for U.S., Canadian, and Australian states or provinces

## Supported Script Deployments

- Server-side scripts cannot deploy on subrecords. Server-side scripts can only access subrecords through the parent record. To accomplish this, a server-side script must be deployed on the parent record. Server-side scripts deployed on the parent record can read, create, edit, and delete address subrecords with the [Subrecord APIs](#).
- Client scripts deployed on the parent record can read and delete address subrecords with the [Subrecord APIs](#).
- Client scripts deployed on the address subrecord can read and edit address subrecords with the standard APIs.

## Important Items to Note Before Scripting on Address Subrecords

- One of the most important things to remember when scripting address subrecords is that the country field determines which address form is used. If your script runs in dynamic mode, you must set the country field first.

- Validate Field, Field Changed, and Post Sourcing events on address fields will not fire in client scripts deployed to Entity or Item Fulfillment records or in custom code on forms for these record types. Instead, add this code to the Custom Code subtab of the address form for the record type.
- You cannot use nlapiGetLineItemValue and nlapiSetLineItemValue to access address fields in dynamic mode. Use nlapiGetCurrentLineItemValue or nlapiSetCurrentLineItemValue instead.
- You cannot use nlobjRecord.getText, nlobjRecord.setFieldText, nlapiGetFieldText and nlapiSetFieldText when scripting subrecords. Use nlobjRecord.getFieldValue, nlobjRecord.SetFieldValue, nlapiGetFieldValue, or nlapiSetFieldValue instead.
- You cannot use nlobjRecord.getLineItemField, nlobjRecord.getField, and nlapiGetField to get address field metadata. You must access the subrecord with the subrecord APIs to get this information.
- You cannot use the subrecord APIs to access address fields on the Company Information page. Access these fields with nlapiLoadConfiguration the same way you would access non-subrecord fields. See [nlapiLoadConfiguration\(type\)](#) for an example.
- If you allow third-party input of address information, you need to translate third-party state input to validated NetSuite state input. For example, if the user enters a state of California, it must be converted to CA.

## Sample Scripts for Address Subrecords

### Creating an Address Subrecord for a New Employee

```
function createEmployee()
{
    var record = nlapiCreateRecord('employee', {recordmode: 'dynamic'});
    record.setFieldValue('companyname','Lead Company 123');
    record.setFieldValue('firstname', 'Lead Company');
    record.setFieldValue('lastname', '123');
    record.setFieldValue('subsidiary','1'); //PARENT COMPANY

    //Add first line to sublist
    record.selectNewLineItem('addressbook');
    record.setCurrentLineItemValue('addressbook', 'defaultshipping', 'T'); //This field is not
    a subrecord field.
    record.setCurrentLineItemValue('addressbook', 'defaultbilling', 'T'); //This field is not
    a subrecord field.
    record.setCurrentLineItemValue('addressbook', 'label', 'First Address Label'); //This fiel
    d is not a subrecord field.
    record.setCurrentLineItemValue('addressbook', 'isresidential', 'F'); //This field is not
    a subrecord field.

    //create address subrecord
    var subrecord = record.createCurrentLineItemSubrecord('addressbook', 'addressbookaddress');

    //set subrecord fields
    subrecord.setFieldValue('country', 'US'); //Country must be set before setting the other ad
    dress fields
    subrecord.setFieldValue('attention', 'John Taylor');
    subrecord.setFieldValue('addressee', 'NetSuite Inc.');
    subrecord.setFieldValue('addrphone', '(123)456-7890');
```

```
subrecord.setFieldValue('addr1', '2955 Campus Drive');
subrecord.setFieldValue('addr2', 'Suite - 100');
subrecord.setFieldValue('city', 'San Mateo');
subrecord.setFieldValue('dropdownstate', 'CA');
    // if the address is not in U.S., Canada, or Australia, use
    // state instead of dropdownstate. For example,
    // subrecord.setFieldValue('state', 'BY');
    // for Bavaria, Germany
subrecord.setFieldValue('zip', '94403');

//commit subrecord and line item
subrecord.commit();
record.commitLineItem('addressbook');

//submit record
var x = nlapiSubmitRecord(record);
}
```

## Accessing Address Subrecord fields on an Existing Customer Record

```
var record = nlapiLoadRecord('customer', 143,{recordmode: 'dynamic'});

record.selectLineItem('addressbook', 2);

var subrecord = record.viewCurrentLineItemSubrecord('addressbook', 'addressbookaddress');

var country = subrecord.getFieldValue('country');
var attention = subrecord.getFieldValue('attention');
```

## Editing an Address Subrecord on an Existing Customer Record

```
var record = nlapiLoadRecord('customer', 143,{recordmode: 'dynamic'});

record.selectLineItem('addressbook', 2);

var subrecord = record.editCurrentLineItemSubrecord('addressbook', 'addressbookaddress');

subrecord.setFieldValue('attention', 'Accounts Payable');

subrecord.commit();
record.commitLineItem('addressbook');

var x = nlapiSubmitRecord(record);
```

## Removing an Address Subrecord on an Employee Record

```
var record = nlapiLoadRecord('employee', 234, {recordmode: 'dynamic'});

record.selectLineItem('addressbook', 3);
record.removeCurrentLineItemSubrecord('addressbook', 'addressbookaddress');
```

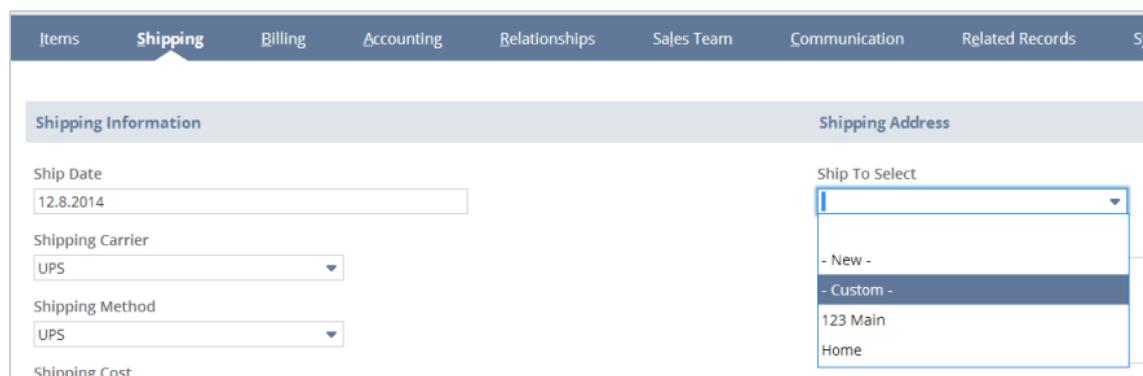
```
record.commitLineItem('addressbook');

var x = nlapiSubmitRecord(record);
```

## Scripting Custom Billing and Shipping Addresses

You can use SuiteScript to create a new address subrecord for an entity and then use that address for custom billing and shipping addresses on transactions. This is useful in situations where the bill-to address is standard, but the ship-to address is a one-time deviation from the norm (or vice versa). For example, this situation can arise with resellers who want to use their own address as the bill-to address, but want to use their customer's address as the ship-to address.

In the UI, custom billing and shipping addresses are created by going to a sales order, clicking either the **Shipping** or **Billing** subtab, and selecting **Custom** from the **Ship To Select** or **Bill To Select** dropdown fields.



The Custom Address popup that displays is a subrecord.

Custom billing and shipping addresses are created and accessed from a body field on the parent record. See [Creating and Accessing Subrecords from a Body Field](#) for additional information on scripting subrecords from a body field. Like other address subrecords, you must set the country first. The primary difference is that the custom billing address and custom shipping address subrecords have their own Internal ID names:

- Billing Address – ‘billingaddress’
- Shipping Address – ‘shippingaddress’

## Examples

For additional script examples of subrecords accessed from a body field, see [Using SuiteScript with Advanced Bin / Numbered Inventory Management](#).

### Creating a New Custom Billing Address

The following server-side code creates a new custom billing address subrecord on an entity.

```
var customerid = 87;

// Open an existing customer record in dynamic mode
var record = nlapiLoadRecord('customer', customerid, {recordmode: 'dynamic'});
```

```
// Create a new address for the customer
var addrSubrecord = record.createCurrentLineItemSubrecord('addressbook', 'addressbookaddress');

//Set the appropriate address subrecord fields
addrSubrecord.setFieldValue('country', 'US');
addrSubrecord.setFieldValue('isresidential', 'F');
addrSubrecord.setFieldValue('attention', 'Billing Address');
addrSubrecord.setFieldValue('addressee', 'NetSuite Inc.');
addrSubrecord.setFieldValue('addrphone', '(123)456-7890');
addrSubrecord.setFieldValue('addr1', '2955 Campus Drive');
addrSubrecord.setFieldValue('addr2', 'Suite - 100');
addrSubrecord.setFieldValue('city', 'San Mateo');
addrSubrecord.setFieldValue('state', 'CA');
addrSubrecord.setFieldValue('zip', '94403');

// Commit the new address subrecord for the customer
addrSubrecord.commit();
record.commitLineitem('addressbook');

// Update the customer record
nlapiSubmitRecord(record);
```

## Creating a New Custom Shipping Address

The following server-side code searches for a specific address subrecord on an entity and then assigns that address to a sales order record.

```
var customerid = 87;
var itemid = 545;
var addressid = -1;

// Load the customer record for read access
var readrecord = nlapiLoadRecord('customer', customerid);

// Get the line item value of the address you want
for(var x = 1; x < readrecord.getLineItemCount('addressbook'); x++) {
    if (readrecord.getLineItemValue('addressbook', 'addressee', x) === 'NetSuite Inc.') {
        addressid = x;
        break;
    }
}

// Create a new sales order record
var record = nlapiCreateRecord('salesorder');

// Set the customer (entity) to the Customer ID
record.setFieldValue('entity', customerid);

// Set the billing and shipping addresses to the line item value that was retrieved
record.setFieldValue('billaddress', addressid);
```

```
record.setFieldValue('shipaddress', addressid);

// Create a new item for the sales order
record.selectNewLineItem('item');

// Set the appropriate fields for the item
record.setCurrentLineItemValue('item', 'item', itemid);
record.setCurrentLineItemValue('item', 'quantity', 1);
record.setCurrentLineItemValue('item', 'location', 1);
record.setCurrentLineItemValue('item', 'amount', '19.99');

// Commit the new item for the sales order
record.commitLineItem('item');

// Update the sales order
nlapiSubmitRecord(record);
```

# Working with Fields

The following topics are covered in this section. If you are new to SuiteScript, they should be read in order:

- Working with Fields Overview
- Referencing Fields in SuiteScript
- Working with Custom Fields in SuiteScript

## Working with Fields Overview

The SuiteScript API includes several [Field APIs](#) you can use to set and get values for built-in NetSuite **standard** fields, as well as for **custom** fields. Standard fields are those that come with NetSuite. Custom fields are those that have been created by NetSuite users to customize their accounts. Custom fields are created using SuiteBuilder point-and-click customization tools.



**Note:** For information on working with nlobjField objects that you can add dynamically to NetSuite records at runtime, see [nlobjField](#) in the NetSuite Help Center. These are the only type of fields you can programmatically add to a record. There are no SuiteScript APIs available for creating custom fields that are akin to the kinds of custom field created using SuiteBuilder point-and-click functionality.



**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.

The following figure shows a combination of body and sublist fields. The body sections of a record include the top (header) portion and non-sublist fields that sometimes appear on the top area of a subtab. Body fields that appear under a subtab should not be confused with sublist fields. Each line on a sublist is referred to as a line item; the fields on each line item are sublist fields. Sublist fields are accessed using [Sublist APIs](#).

On the figure below:

1. **Body fields** - can be a mix of standard and custom fields.
2. **Sublist fields** - fields on a sublist. See [Working with Subtabs and Sublists](#) for more information.

**Sales Order**  **PENDING BILLING**

**APPROVAL STATUS:**

**Primary Information**

CUSTOM FORM *	Standard Sales Order	END DATE	<input type="text"/>
ORDER #	SORD100491	PO #	<input type="text"/>
CUSTOMER *	AAKASH CHEMICAL	MEMO	<input type="text"/>
PROJECT	16 Billing project		
DATE *	31.7.2013		
STATUS *	Pending Billing		
START DATE	<input type="text"/>		

**Sales Information**

OPPORTUNITY	<input type="text"/>	<input type="checkbox"/> EXCLUDE COMMISSIONS	LEAD SOURCE	<input type="text"/>
SALES EFFECTIVE DATE	31.7.2013			

**Classification**

DEPARTMENT	CLASS	LOCATION	East Coast
DEFERRED REVENUE RECLASSIFICATION ACCOUNT	READY FOR SHIPMENT	LAST MODIFIED DATE	31.7.2013
FOREIGN CURRENCY ADJUSTMENT REVENUE ACCOUNT	SALES REP PHONE	CREATED DATE	31.7.2013
CUSTOMER CATEGORY	POINTS	<input type="text"/> 0	

**Items**

<input type="checkbox"/> ENABLE ITEM LINE SHIPPING	DISCOUNT ITEM	<input type="text"/>
COUPON CODE	RATE	<input type="text"/>
PROMOTION	<input type="button" value="Calculate"/>	<input type="text"/>
<input type="button" value="Add Multiple"/> <input type="button" value="Upsell Items"/> <input type="button" value="Close Remaining Lines"/> <input type="button" value="Refresh Items from Project"/> <input type="button" value="Clear All Lines"/>		

**ITEM \***

Software Service 202	1
Software Service 401	1

## Referencing Fields in SuiteScript

Many SuiteScript APIs allow you to get, set, or search for the value of a particular field. Whether you are referencing a standard field or a custom field, when you reference the field in SuiteScript, you will use the field's internal ID. To obtain field internal IDs, see [How do I find a field's internal ID?](#) in the NetSuite Help Center.



**Important:** Be aware that not every field that appears in your NetSuite account officially supports SuiteScript. To write scripts that include only supported, officially tested NetSuite fields, it is recommended you refer to the [SuiteScript Records Browser](#) to verify a field's official support. See [SuiteScript Reference](#) for more details.

## Getting Field Values in SuiteScript

If you are using SuiteScript to process record data in **standard** mode (as opposed to **dynamic** mode), be aware of the following when using “getter” APIs to get the value of a field:



**Note:** If you are not familiar with standard mode and dynamic mode scripting, see the help topic [Working with Records in SuiteScript](#) in the NetSuite Help Center.

To check if a field has a non-empty value, NetSuite recommends that you write code which checks for null and empty when using any of the following APIs:

- `nlapiGetFieldValue(fldnam)`
- `nlapiGetLineItemValue(type, fldnam, linenum)`
- `nlobjRecord.getFieldValue(name)`
- `nlobjRecord.getLineItemValue(group, name, linenum)`

For guidance and examples of syntax for scripting with fields, click on one of the links above to view details for each specific API.



**Important:** Note that this inconsistency in field return values does NOT exist when scripting records in **dynamic** mode.

The following snippet provides an example of how you might want to write your code to catch any null vs. empty string return value inconsistencies:

```
if (value)
{
    // handle case where value is not empty
}
-or-
if (!value)
{
    // handle case where value is empty (or null)
}
```

## Working with Custom Fields in SuiteScript

You can use SuiteScript APIs to get, set, and search the values of custom fields that have been created using SuiteBuilder. Note, however, you can only set the value of custom fields that have a **stored value**. This follows the behavior of the UI.

The following figure shows a custom entity field. The field's UI label is Contact Source and its internal ID is custentity11. In this figure, the **Store Value** check box is selected, which means that you can use SuiteScript to get and set the value of this custom entity field.

**Custom Entity Field**

**LABEL:** \* Contact Source

**ID:** custentity11

**INTERNAL ID:** 33

**OWNER:** K Wolfe

**DESCRIPTION:**

**TYPE:** Multiple Select

**LIST/RECORD:** Campaign

STORE VALUE  USE ENCRYPTED FORMAT

When a custom field does not contain a stored value (the Store Value check box is not selected), you *can* reference this field in a SuiteScript search to return the current value of the field. However, non-stored custom fields are considered to have dynamic values, so in a search, the value of a non-stored custom field might be 10 one day and 12 the next day when the same search is executed.

**Note:** If you are not familiar with creating custom fields in NetSuite, see the help topic [Custom Fields](#) in the NetSuite Help Center.

## Providing Internal IDs for Custom Fields

If you are using SuiteBuilder to create a custom field, and you plan to reference the field in your scripts, NetSuite recommends you create an internal ID that includes an underscore ( `_` ) after the custom field's prefix. You should then add a meaningful name after the underscore. This will enhance readability in your SuiteScript code.

For example, if you are using SuiteBuilder to create a custom transaction body field with the UI label Contact Fax, the field's internal ID should be something equivalent to `custbody_contactfax`. Note that you do not need to write the custom field's prefix in the ID field (see figure below). After the custom field definition is saved, the prefix for that custom field type is automatically added to the ID. When the custom transaction body field (below) is saved, its internal ID will appear as `custbody_contactfax`. This is the ID you will reference in your scripts.

**Transaction Body Field**

**LABEL:** \* Contact Fax

**ID:** custbody\_contactfax

**INTERNAL ID:** 15

**DESCRIF:**

**TYPE:** Phone

## Understanding Custom Field Prefixes

As a reference, the following table provides the prefixes for each custom field type. You do not need to type these prefixes when you assign an internal ID to a custom field. This table is provided only for convenience to SuiteScript developers who may be working with different custom field types and are not sure how to identify the field type using the prefix.

Custom field type	Custom field prefix
Entity field	custentity
Item field	custitem
CRM field	custevent
Transaction body field	custbody
Transaction column field	custcol
Transaction item options	custcol
Item number fields	custitemnumber
Other custom fields	custrecord

# Working with Subtabs and Sublists

- Subtabs and Sublists Overview
- Subtabs and Sublists - What's the Difference?
- Sublist Types

## Subtabs and Sublists Overview

When using SuiteScript on subtabs and sublists, you should be aware of the following:

1. The distinction between subtabs and sublists (see [Subtabs and Sublists - What's the Difference?](#))
2. Sublist types (see [Sublist Types](#))
3. Adding subtabs with SuiteScript ([Adding Subtabs with SuiteScript](#))
4. Adding sublists with SuiteScript ([Adding Sublists with SuiteScript](#))
5. Manipulating sublist with SuiteScript ([Working with Sublist Line Items](#))
6. Sublist scripting when a record is in dynamic mode ([Working with Sublists in Dynamic Mode and Client SuiteScript](#))

**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.

**Note:** For a list of all sublists that support SuiteScript, see [Scriptable Sublists in the NetSuite Help Center](#). To see all sublist-related APIs, see [Sublist APIs](#).

## Subtabs and Sublists - What's the Difference?

Subtabs and sublists both look like tabs in the UI (see figure). However, functionally they serve very different purposes. See these sections to learn about the differences between subtabs and sublists:

- [What is a Subtab?](#)
- [What is a Sublist?](#)

The screenshot shows a NetSuite page with a top navigation bar containing tabs: Relationships, Communication, Address, Sales (highlighted with a red box), Marketing, Support, Financial, Preferences, System Information, Custom, Special Instructions, and Satisfaction Surveys. A green circle with the number 1 is positioned above the Sales tab. Below the tabs is a dropdown menu labeled 'TERRITORY' with a green circle containing the number 2. The main content area contains a subtab titled 'Sales Team' (highlighted with a red box) which lists 'Opportunities', 'Transactions', 'Items Purchased', 'Upsell', 'Projects', and 'Qualification'. A green circle with the number 3 is positioned above the subtab content. Below the subtab is a table with columns: 'EMPLOYEE\*', 'SALES ROLE\*', 'PRIMARY', and 'CONTRIBUTION %\*'. The table lists five employees: Clark Koozer, Krista Barton, Neil Thomson, Mark Grogan, and Sam R Cruz, each assigned as a 'Sales Rep' with a '20.0%' contribution rate. At the bottom of the table are buttons for 'Add', 'Cancel', 'Insert', and 'Remove'.

1	Parent Subtab
2	Child Subtab
3	Sublist

## What is a Subtab?

Subtabs contain body fields, other subtabs, and sublists. Unlike sublists, subtabs do not contain references to other records. Subtabs are used mainly for organizational purposes.

The figure below shows the Sales subtab on a Customer record. Notice that the Sales tab contains body fields that hold data specific to the Customer. The primary purpose of the Sales subtab is to organize all of the sales-related sublists (Sales Team, Opportunities, Transactions, and so on).

EMPLOYEE	SALES ROLE	PRIMARY	CONTRIBUTION %
Clark Koozer	Sales Rep		20.0%
Krista Barton	Sales Rep		20.0%
Neil Thomson	Sales Rep		20.0%
Mark Grogan	Sales Rep		20.0%
Sam R Cruz	Sales Rep		20.0%

To compare what you see on the Sales subtab, the Sales Team sublist contains data that link to other records—in this case, the employee records for the sales people associated with this customer (see figure).

EMPLOYEE*	SALES ROLE*	PRIMARY	CONTRIBUTION %*
Clark Koozer	Sales Rep		20.0%
Krista Barton	Sales Rep		20.0%
Neil Thomson	Sales Rep		20.0%
Mark Grogan	Sales Rep		20.0%
Sam R Cruz	Sales Rep		20.0%

1	Child Subtab
2	Sublist

The next figure shows the Financial subtab, also on the Customer record. Notice that the information on this subtab is additional field-level information related to this particular customer. None of the information applies to or references data that exists on another record.

In SuiteScript you can access fields that appear on a subtab using [Field APIs](#). Field APIs are also used on regular body fields that appear on the top portion of records.

DATE	ITEM	PAYROLL ITEM	DURATION	APPROVED	STATUS	TYPE
9/18/2014	Hardware Repair (on-site)		4:00	No	Unbilled	Actual Time
9/18/2014	Labor 1		2:00	No	Unbilled	Actual Time

## What is a Sublist?

Sublists contain a list of references to other records. Note that the list of record references are referred to as **line items**. Within NetSuite there are four types of sublists: editor, inline editor, list, and static list (see [Sublist Types](#) for details on each type).



**Important:** Static list sublists do not support SuiteScript. For a list of all editor, inline editor, and list sublists that support SuiteScript, see [Scriptable Sublists](#) in the NetSuite Help Center.

The following figure shows the [Item Pricing Sublist](#) on the Customer record. This is an **inline editor** sublist that appears on a subtab, in this case the Financial subtab. Whereas the field-level data captured on the Financial subtab applies specifically to this customer, the data on the Item Pricing sublist references data contained on other records.

In the UI, you can add/insert/remove lines items to this sublist using the Add, Insert, and Remove buttons. In SuiteScript, you can perform the same actions using [Sublist APIs](#) such as `nlapilInsertLineItem(type, line)` and `nlapiremoveLineItem(type, line)`.

The screenshot shows a NetSuite account setup page. At the top, there are tabs: Relationships, Communication, Address, Sales, Marketing, Support, **Financial**, Preferences, System Information, and Custom. A green circle labeled '1' is over the Financial tab. Below the tabs are sections for Account Information, Tax Information, and Balance Information. In the Tax Information section, there is a checkbox labeled 'TAXABLE' with a checked mark. A green circle labeled '2' is over this checkbox. In the Balance Information section, there is a table with columns: BALANCE, OVERDUE BALANCE, and DAYS OVERDUE. The values are 80.00, 209.87, and 4,097 respectively. A green circle labeled '3' is over the 'Item Pricing' subtab. The 'Item Pricing' subtab is active, showing a list of items: Autopsy Saws, Assorted Bandages, Sterile, Large, Blue, and OR Equipment : IMED Gemini PC1 IV Pump. Each item has a dropdown menu next to it and a 'PRICE LEVEL' column with options like Alternate Price 3, Alternate Price 2, Base Price, and Base Price. Below the list are buttons for Add, Cancel, Insert, Remove, Save, Cancel, and Reset.

1	Parent Subtab
2	Child Subtab
3	Sublist

## Sublist Types

There are four types of sublists in NetSuite:

- Editor Sublists
- Inline Editor Sublists
- List Sublists
- Static List Sublists



**Important:** Static list sublists do not support SuiteScript. Scripts written against static list sublists will either not run or will return a system error. All other sublist types support both client and server SuiteScript.



**Note:** If you are building your own custom form and are adding a sublist object to that form through `nlobjForm.addSubList(name, type, label, tab)`, you can set the sublist `type` to any of the four sublist types. You can then write scripts against your custom sublist. Note that sorting (in the UI) is not supported on static sublists created using the `addSubList(...)` method if the row count exceeds 25.

## Editor Sublists

The editor sublist allows users to insert/edit/remove lines dynamically prior to submitting the form. On an editor sublist, editing sublists lines (referred to as line items) is done in fields directly above the line items. In the UI, changes you make when you add/edit/remove lines are not committed to the database until you save the entire record. Similarly, in SuiteScript add/edit/remove functions provided in [Sublist APIs](#) are not persisted in the NetSuite database until the change is committed to the NetSuite database.

When writing client scripts, you must call `nlapiCommitLineItem(type)` after each sublist line change. Otherwise your changes will not be committed to NetSuite.

When writing server scripts, you must call `nlobjRecord.commitLineItem(group, ignoreRecalc)` to commit sublist updates. Note that you must do this in addition to calling `nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)`, which commits the entire record object to the database.



**Note:** In SuiteScript, the first sublist line item is numbered 1, not 0.

## Inline Editor Sublists

Inline editor sublists are similar to [Editor Sublists](#) in these ways:

- you can add/edit/remove lines dynamically prior to submitting the form
- you can add/edit/remove lines using the UI or SuiteScript
- when writing client scripts, you must call `nlapiCommitLineItem(type)` after each sublist line change. Otherwise your changes will not be committed to NetSuite.
- When writing server scripts, you must call `nlobjRecord.commitLineItem(group, ignoreRecalc)` to commit sublist updates. Note that you must do this in addition to calling `nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)`, which commits the entire record object to the database.

The **only** difference between an inline editor sublist and an editor sublist is UI appearance. Inline editor sublists do not contain a line item edit area directly above the line items. The line items on an inline editor sublist are edited “inline” directly on the lines on which they appear.

The following figure shows the [Items Sublist](#) on the Estimate record. The field-level data that appears directly above the line items are not used for adding, editing, or removing line items that appear below it.

In SuiteScript, fields above the line items are accessed using [Field APIs](#). Sublist line items are accessed using [Sublist APIs](#).

**Note:** In SuiteScript, the first sublist line item is numbered 1, not 0.

The screenshot shows a SuiteScript sublist editor. At the top, there are tabs for Items, Shipping, Billing, Accounting, Relationships, Sales Team, Communication, Address, Custom, and Warranty. Under the Items tab, there are checkboxes for 'ENABLE ITEM LINE SHIPPING' and 'COUPON CODE'. Below these are sections for 'PROMOTION' and 'Calculate'. Buttons for 'Add Multiple', 'Upsell Items', and 'Clear All Lines' are also present. The main area displays a table with columns: ITEM \*, QUANTITY, UNITS, INVENTORY DETAIL, DESCRIPTION, PRICE LEVEL, RATE, AMOUNT, OPTIONS, EXPECTED SHIP DATE, ALT. SALES, and SHIP TO. A row for 'Assorted Bandages - Medium - Blue' has a quantity of 3 selected. To the right of this row, there are discount and rate fields: 'DISCOUNT' (5% discount) and 'RATE' (-5.0%). Below the table are buttons for 'OK', 'Cancel', 'Make Copy', 'Insert', and 'Remove'. A summary row at the bottom shows 'Assorted Bandages - Large - Blue' with a quantity of 3, a description of 'Assorted Large Bandages', a price level of 'Base Price', a rate of 74.00, and a total amount of 222.00.

## List Sublists

Unlike [Editor Sublists](#) and [Inline Editor Sublists](#), **list** sublists are not dynamic. The number of line items are fixed and cannot be removed or added on-the-fly though UI customization or through SuiteScript.

Changes you make to existing line items on list sublists are submitted along with the main record and do not take effect until after the record has been saved. Note that even though you cannot add or remove lines on a list sublist, you can use the UI to change values or SuiteScript to get/set values on lines that currently exist.

In SuiteScript you would not use [Sublist APIs](#) such as `nlapiSelectNewLineItem(type)`, `nlapiInsertLineItem(type, line)`, or `nlapiRemoveLineItem(type, line)` to add or remove line items. Neither will you use the `nlapiCommitLineItem(type)` or `nlapiRefreshLineItems(type)` APIs in the context of a list sublist.

Also note that in SuiteScript, client lineInit and validateLine functions will not execute, since they have no context in a list sublist. (For information on client event functions, see [Client Event Types](#).)

The following figure shows the Subscriptions child sublist on the Customer record. Although you cannot add/remove lines, you can edit the lines that are there (in this case, you can select or de-select the check boxes).

**Note:** In SuiteScript, the first sublist line item is numbered 1, not 0.

The screenshot shows a SuiteScript sublist editor for the 'Subscriptions' tab. At the top, there are tabs for Campaigns, Subscriptions\*, Keywords, Click-Streams, Page Hits, Hosted Page Hits, Referrer, and Cart Conte. Below these are dropdown menus for 'GLOBAL SUBSCRIPTION STATUS' (set to 'Soft Opt-In') and 'SOFT OPT-IN' (set to 'Opt-In'). The main area displays a table with columns: SUBSCRIBED, SUBSCRIPTION, and LAST MODIFIED. There are five rows, each with a checked checkbox in the 'SUBSCRIBED' column. The subscriptions listed are 'Billing Communication', 'Marketing', 'Newsletters', 'Product Updates', and 'Surveys', all modified on 25.9.2014 2:43 pm.

The next figure shows the Apply sublist on the Accept Customer Payments record. Similar to the Subscriptions sublist, you can manipulate the line items that appear, but you cannot dynamically add or remove additional lines.

APPLY	DATE	PROJECT/SUB	TYPE	REF NO.	ORIG. AMT.	AMT. DUE	CURRENCY	DISC. DATE	DISC. AVAIL.	DISC. TAKEN	PAYMENT
<input checked="" type="checkbox"/>	27.2.2013		Invoice	INVC10511	250.00	250.00	USA				250.00
<input checked="" type="checkbox"/>	28.2.2013		Invoice	INVC10512	600.00	600.00	USA				600.00
<input checked="" type="checkbox"/>	13.8.2013		Invoice	INVC10538	7,350.00	7,350.00	USA	23.8.2013	147.00		7,350.00
<input checked="" type="checkbox"/>	27.8.2013		Invoice	INVC10539	6,752.50	6,752.50	USA	6.9.2013	135.05		6,752.50
<input type="checkbox"/>	15.2.2013		Invoice	INVC10540	70.14	70.14	USA				
<input type="checkbox"/>	15.2.2013		Invoice	INVC10541	45.00	45.00	USA				

The last figure provides another example of a list sublist—the **Billable Expenses Sublist** on the Invoice record. Again, you can only manipulate the line item data provided. You cannot dynamically add or remove items. Any changes you make to the data on this sublist will not be committed to the database until you call `nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)` in your script.

APPLY	DATE	EMPLOYEE	CATEGORY	MEMO	ORIGINAL AMOUNT	BILL AMOUNT	TAX CODE
<input checked="" type="checkbox"/>	20.8.2008	Leaf, Vicky	Travel	Travel: mileage	80.00	80.00	-Not Taxable-

## Static List Sublists



**Important:** SuiteScript is not currently supported on static list sublists.

Static list sublists, also referred to as read-only sublists, contain static data. These sublists are typically used for displaying associated records/data rather than child records/data. Technically, this means that the data on a static list sublist is not actually part of the record (and therefore not accessible to SuiteScript), and is not submitted with the record when the record is saved.

The following figure shows the System Notes sublist, which is accessed through the System Information subtab on many records. Note that all data in the System Notes sublist is read-only and is not even settable through the UI.

DATE	SET BY	CONTEXT	TYPE	FIELD	OLD VALUE	NEW VALUE
25.9.2014 2:43 pm	Patek, T	UI	Set	Global Subscription Status		Soft Opt-In
25.9.2014 2:43 pm	Patek, T	UI	Set	Territory		Default Round-Robin
25.9.2014 2:43 pm	Patek, T	UI	Set	Customer ID		70
25.9.2014 2:43 pm	Patek, T	UI	Set	Budget Approved		F

The next figure shows the Files sublist, which is accessed from the Communications subtab on many records. In this case you can attach/detach a file to this sublist, but the file is maintained entirely as a separate document. The data in this document (in this case a .txt file), is not considered to be part of Customer record, which can be manipulated through the UI or through SuiteScript.

ATTACHED FILES	FOLDER	SIZE (KB)	LAST MODIFIED	DOCUMENT TYPE	REMOVE	EDIT	DOWNLOAD
sample file.txt	SuiteScripts	1	2.3.2009 2:37 pm	Plain Text File	Remove	Edit	download

The last figure shows the User Notes sublist, also accessed through the Communication subtab. Although you can add a new Note to this sublist, the data you define on the actual Note record is not available to this Customer record. Therefore, the User Notes sublist is considered to hold static/read-only data.

EDIT	DATE	AUTHOR	TITLE	MEMO
Edit	6.11.2014 10:52 am	Patek, T	Email Customer	Notify customer they are late with their payment.

**Note:** In some cases you *can* use search joins in SuiteScript to search the data on a static list sublist (for example, data related to notes, contacts, messages, or files that appear on a particular record). In the previous example, you could use the `file` search join to search for all files associated with this particular Customer record.

## Adding Subtabs with SuiteScript

You can add subtabs to custom forms through UI point-and-click customization and through SuiteScript. In scripting, you must use either of the following two `nlobjForm` methods, depending on your use case:

- `addTab(name, label)` — to add a top-level tab
- `addSubTab(name, label, tab)` — to create a nested subtab



**Important:** You must define **two** subtabs for subtab UI labels to appear. If you define only one subtab in your script, the UI label you provide for the subtab will not actually appear in the UI.

Both methods return an `nlobjTab` object, through which you can further define the properties of your tab.



**Note:** To add subtabs using UI customization, in the NetSuite Help Center, see the help topics [Adding Subtabs to a Custom Record](#) and [Configuring Subtabs for Custom Entry and Transaction Forms](#).

## Example

The following example shows how to use SuiteScript to add subtabs to a custom NetSuite form. This script is a `beforeLoad` user event script that is deployed to the Sales Order. Note that 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.

When you are adding [UI Objects](#) to an existing form, be sure to prefix the internal IDs for all elements with `custpage`, for example '`custpage_sample_tab`' and '`custpage_field_email`' (see sample). In the sample below, the `nlobjTab` and `nlobjField` UI objects are being added to a custom transaction form on a Sales Order. (See the help topic [Custom Transaction Forms](#) in the NetSuite Help Center if you are not familiar with this form type.)

Also note that element internal IDs must be in all lowercase.

```
//Define the user event beforeLoad function
function tabsToSalesOrder(type, form)
{
//Define the values of the beforeLoad type argument
if (type == 'create')
{
//Add a new tab to the form
var sampleTab = form.addTab('custpage_sample_tab', 'Sample Tab');

//Add a field to the new tab
var newFieldEmail = form.addField('custpage_field_email', 'email', 'Alt Email', null,
'custpage_sample_tab');

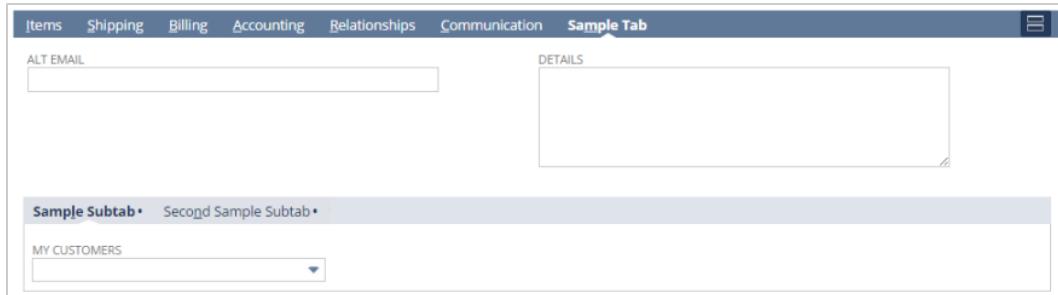
//Add a second field to the new tab
var newFieldText = form.addField('custpage_field_text', 'textarea', 'Details', null,
'custpage_sample_tab');

//Add a subtab to the first tab
var sampleSubTab = form.addSubTab('custpage_sample_subtab', 'Sample Subtab',
'custpage_sample_tab');

//Add a select field to the subtab
var newSubField = form.addField('custpage_sample_field', 'select', 'My Customers', 'custome
r',
'custpage_sample_subtab');
```

```
//Add a second subtab to the first tab
var sampleSubTab = form.addSubTab('custpage_sample_subtab2', 'Second Sample Subtab',
'custpage_sample_tab');

//Add a field to the second subtab
var newSubField = form.addField('custpage_sample_field2', 'select', 'My Employees', 'employ
ee',
'custpage_sample_subtab2');
}
```



## Adding Sublists with SuiteScript

You can add sublists to custom forms through UI point-and-click customization and through SuiteScript. In scripting, you must use the `nlobjForm.addSubList(name, type, label, tab)` method to add a sublist. This method returns an `nlobjSubList` object, through which you can further define the properties of your sublist.



**Important:** The internal ID for all custom sublists, subtabs, and fields must be prefixed with `custpage`. The rest of the ID name must be in lowercase.

When adding a sublist through scripting you must define:

1. The custom sublist internal ID.

**Example :** ' custpage \_contacts'

2. The sublist type you are defining.

**Example :** 'editor', 'inlineeditor', 'list', or 'staticlist'

3. The UI label for the sublist.

**Example :** 'Custom Contacts'

4. The subtab on which the sublist will appear.

**Example :** 'general' or 'custpage\_mynewsTAB'



**Important:** To add sublists through point-and-click customization, see the help topic [Custom Sublists](#) in the NetSuite Help Center. When adding a sublist through UI customization, you are essentially adding the data from a saved search, the results of which are only associated with the record. This is the equivalent of a static list sublist. The results are not considered to be part of the actual record.

## Example 1

This sample shows how to create a custom sublist and run a search every time the form is loaded, edited, or viewed. This is a beforeLoad user event script.

```
function beforeLoadSublist(type, form)
{
    if (type=='edit' || 'view')
    {
        //add a sublist to the form. Specify an internal ID for the sublist,
        //a sublist type, sublist UI label, and the tab the sublist will appear on
        var contacts = form.addSubList('custpage_contacts', 'staticlist', 'Custom Contacts', 'general');

        //add fields to the sublist
        contacts.addField('entityid', 'text', 'Name');
        contacts.addField('phone', 'phone', 'Phone');
        contacts.addField('email', 'email', 'Email');

        // perform a Contact record search. Set search filters and return columns for
        // the Contact search
        var contactdata = nlapiSearchRecord('contact', null, new
            nlobjSearchFilter('company', null, 'anyOf', nlapiGetRecordId()),
            [new nlobjSearchColumn('entityid'), new nlobjSearchColumn('phone'),
            new nlobjSearchColumn('email')])

        // display the search results on the Custom Contact sublist
        contacts.setLineItemValues(contactdata)
    }
}
```

## Example 2

The following example shows how to add an inline editor sublist to a Suitelet by instantiating the nlobjForm object (using [nlapiCreateForm\(title, hideNavbar\)](#)) and then calling [nlobjForm.addSubList\(name, type, label, tab\)](#). Note that because you are not adding field or sublist elements to an existing NetSuite form, you do not need to prefix the element internal IDs with `custpage`.

## Script:

```
function createSuiteletWithSublist(request, response)
{
    if (request.getMethod() == 'GET' )
    {
```

```
// create the form
var form = nlapiCreateForm('Simple Form');

// add fields to the form
var field = form.addField('textfield','text', 'Text');
field.setLayoutType('normal','startcol')
form.addField('datefield','date', 'Date');
form.addField('currencyfield','currency', 'Currency');
form.addField('textareafield','textarea', 'Textarea');

// add a select field and then add the select options that will appear in the dropdown
var select = form.addField('selectfield','select','Custom');
select.addSelectOption("", "");
select.addSelectOption('a','Albert');
select.addSelectOption('b','Baron');
select.addSelectOption('c','Chris');
select.addSelectOption('d','Drake');
select.addSelectOption('e','Edgar');

// add a sublist to the form
var sublist = form.addSubList('sublist','inlineeditor','Inline Editor Sublist', 'tab1');

// add fields to the sublist
sublist.addField('sublist1','date', 'Date');
sublist.addField('sublist2','text', 'Name');
sublist.addField('sublist3','currency', 'Currency');
sublist.addField('sublist4','textarea', 'Large Text');
sublist.addField('sublist5','float', 'Float');

// make the Name field unique. Users cannot provide the same value for the Name field.
sublist.setUniqueField('sublist2');

form.addSubmitButton('Submit');

response.writePage( form );
}

}
```

DATE	NAME	CURRENCY	LARGE TEXT	FLOAT
11/4/2014	Jane Doe	10.00		0.978
11/2/2014	John Doe	15.00		0.1357

**i Note:** The `nlapicRefreshLineItems(type)` API can be used to refresh static list sublists that have been added using `nlobjSubList`. This API implements the behavior of the Refresh button on the UI.

## Working with Sublist Line Items

NetSuite provides several [Sublist APIs](#) to manipulate sublist line items. You can use these APIs to add or remove line items, update multiple line items when a body field is changed, or automate the population of line items when certain conditions exist on the form.

When scripting with sublists, you should know if you are scripting an [Editor Sublists](#), [Inline Editor Sublists](#), or [List Sublists](#) sublist. Because [List Sublists](#) sublists are not dynamic, you cannot add/remove lines. You can only get/set values that currently exist on the sublist.

Note, however, whether you are scripting an editor, inline editor, or list sublist, generally you will specify one or more of the following in the sublist API:

1. The sublist internal ID

**Example :** 'salesteam' (appears in the UI as the **Sales Team** sublist)

2. The sublist line item (field) ID.

**Example :** 'isprimary' (appears in the UI as **Primary** )

3. The sublist line number — doing so enables you to specify *where* on the sublist you want to change, add, or remove a line. Note that first line on a sublist is 1 (not 0).

4. The value of the line item.

**Example :** The value can be defined directly in the API or it can be a passed in value that was defined elsewhere in the script.

## Example

```
nlapiSetLineItemValue('salesteam', 'isprimary', 2, 'T');
```

## Adding and Removing Line Items

To add/remove sublist line items, follow the general guidelines provided below. The approach you follow depends on whether you are writing a client script to attach to a record, or a server script that loads a record from the database. (In this context, scripts that are considered to be **server scripts** are Suitelets, user event scripts, and scheduled scripts. Scripts considered to be **client scripts** are form- and record-level client scripts.)



**Important:** This section does not apply to [List Sublists](#). List sublists contain information that cannot be dynamically added or removed in either the UI or in SuiteScript. For information on getting/setting existing values on a list sublist, see [Getting and Setting Line Item Values](#).

### Client Scripts

1. (Optionally) Call [nlapiGetLineItemCount\(type\)](#) to get the number of lines in the sublist. Alternatively you can call [nlapiGetCurrentLineItemIndex\(type\)](#) to return the number of the currently select line item.
2. Call either [nlapiInsertLineItem\(type, line\)](#) or [nlapiRemoveLineItem\(type, line\)](#) to add/remove a line. In the **line** argument you will specify the line number of the line you want to add/remove.
3. If adding a line:
  1. Call [nlapiSelectNewLineItem\(type\)](#) to select and insert a new line (as you would in the UI).
  2. Call [nlapiSetCurrentLineItemValue\(type, fldnam, value, firefieldchanged, synchronous\)](#) to set the value of the line.
4. Call [nlapiCommitLineItem\(type\)](#) to commit/save the changes to the sublist.
5. Perform steps 3 and 4 as many times as necessary to add all line items.

### Server Scripts

1. Load the record object — for example using [nlapiLoadRecord\(type, id, initializeValues\)](#).
2. (Optionally) Call [nlobjRecord.getLineItemCount\(group\)](#) to get the number of lines in the sublist.
3. Call either [nlobjRecord.insertLineItem\(group, linenum, ignoreRecalc\)](#) or [nlobjRecord.removeLineItem\(group, linenum, ignoreRecalc\)](#) to add/remove a line. In the **group** argument specify by line number where to add/remove the line. Line numbering begins with 1, not 0.
4. If adding a line:
  1. Call [nlobjRecord.selectNewLineItem\(group\)](#) to select and insert a new line (as you would in the UI).
  2. Call [nlobjRecord.setCurrentLineItemValue\(group, name, value\)](#) to set the value of the line.
5. Call [nlobjRecord.commitLineItem\(group, ignoreRecalc\)](#) to commit/save the changes to the sublist.
6. Submit the record using [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#).

#### Example 1 (Server Script)

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 `nlobjRecord.selectNewLineItem()` method. You then set all values for the new lines using the `nlobjRecord.setCurrentLineItemValue()` method. When you are finished adding values to each line in the sublist, you must commit each line to the database. You will call the `nlobjRecord.commitLineItem()` method to commit each line.

```
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);
```

This sample shows how to add a line to a sublist. When the record is saved, the updates to the sublist are committed to the database.

```
//Load a sales order. 187 is the internal ID of the sales order
var rec = nlapiLoadRecord('salesorder', 187);

//Insert a new line at the start of Item sublist
rec.insertLineItem('item', 1);

//Set the value of quantity to 10 on the first line of the sublist
rec.setCurrentLineItemValue('item', 'quantity', 1, 10);

//Set the value of currency to 1 (the internal ID for US dollar) on the first line of the sublist
rec.setCurrentLineItemValue('item', 'currency', 1, 1);

//Submit the record to commit the sublist changes to the database
var id = nlapiSubmitRecord(rec, true);
```

## Example 2 (Server Script)

This sample shows how to use `nlobjRecord.getLineItemCount(group)`, which is used to determine the number of lines in a sublist. In this sample, a line item is added to the end of the Items sublist. When the record is saved, the updates to the sublist are committed to the database.

```
//Get the new record
```



```

var rec = nlapiGetNewRecord();

//Determine the number of lines on the Item sublist
var intCount = rec.getLineItemCount('item');

//Insert a line after the line that already exists
rec.insertLineItem('item', intCount + 1);

//Set the value of the line item
rec.setCurrentLineItemValue('item', 'quantity', intCount + 1, 10);

// Commit the sublist line changes
rec.commitLineItem('item');

// Submit the record to commit all change to the database
var id = nlapiSubmitRecord(rec, true);

```

### Example 3 (Client Script)

This sample shows how to add a line item to a transaction using a client script. Be aware that in client scripting you must always use [nlapiCommitLineItem\(type\)](#) to commit any line item changes to the sublist.

In this example you first insert the line and then commit the line. If you set the item field using [nlapiSetCurrentLineItemValue\(type, fldnam, value, firefieldchanged, synchronous\)](#), you cannot call [nlapiCommitLineItem](#) until the server call for the item information has completed. The only way to know that the server call is complete is to create a post-sourcing function that sets a flag.

For example, suppose you want to insert a shipping line when a user clicks a button. You can attach a function such as [insertShippingRate\(\)](#) to that button, which adds an item named "Shipping", sets its rate, and then commits the line.

```

function insertShippingRate()
{
    nlapiSelectNewLineItem('item');

    /* important so that you know that the script was called from insertShippingRate() */
    nlapiSetCurrentLineItemValue('item', 'custcolinsertshippingrate', true);
    nlapiSetCurrentLineItemText('item', 'item', 'Shipping');
}

function doPostSourcing(type, fldname)
{
    if ( type == 'item' && fldname == 'item' && nlapiGetCurrentLineItemValue
        ('item', 'custcolinsertshippingrate') == true )
    {
        nlapiSetCurrentLineItemValue('item', 'custcolinsertshippingrate', false);
        nlapiSetCurrentLineItemValue('item', 'rate', '7.50');
        nlapiCommitLineItem('item');
    }
}

```

# Getting and Setting Line Item Values

You can use both client and server scripts to get/set line item values. The set/get guidelines provided here can be used on [Editor Sublists](#), [Inline Editor Sublists](#), and [List Sublists](#) sublists.

## Example 1

The following sample includes several [Sublist APIs](#). This sample copies sales reps from the Sales Team sublist of one sales order to another sales order, ignoring those on the Sales Team sublist who are not sales reps.

```
// Copy all the reps from the original order to the adjusting order
var iRep = 1;
var reps = originalSo.getLineItemCount('salesteam');

for (var rep = 1; rep <= reps; rep++)
{
    // If the role is not sales rep, ignore it
    if (originalSo.getLineItemValue('salesteam', 'salesrole', rep) != '-2')
        continue;
    var repct = originalSo.getLineItemValue('salesteam', 'contribution', rep);
    if (repct != '0.0%')
    {
        var repId = originalSo.getLineItemValue('salesteam', 'employee', rep);
        // keep the percent the same
        if (repct.substring(repct.length-1) == '%')
        {
            //remove the percent sign % from the end
            repct = repct.substring(0, repct.length-1);
        }
        so.insertLineItem('salesteam', iRep);
        so.setCurrentLineItemValue('salesteam', 'contribution', iRep, repct);
        so.setCurrentLineItemValue('salesteam', 'employee', iRep, repId);

        // copy the role
        so.setCurrentLineItemValue('salesteam','salesrole',iRep,originalSo.getLineItemValue
        ('salesteam','salesrole', rep));

        // If primary rep on original order make it primary on the new sales order
        var primary = originalSo.getLineItemValue('salesteam', 'isprimary', rep);
        so.setCurrentLineItemValue('salesteam', 'isprimary', iRep, primary);
        iRep++;

        so.commitLineItem('salesteam');
    }
}

// save the new order and return the ID
var sold = nlapiSubmitRecord(so, true);
```

## Example 3

The following sample is a validateLine client script which uses [nlapiGetCurrentLineItemValue\(type, fldnam\)](#) to prevent the addition of Sales Order item lines with an amount greater than 10000.

```

function validateLine(group)
{
    var newType = nlapiGetRecordType();
    if ( newType == 'salesorder' && group == 'item' && parseFloat(nlapiGetCurrentLineItemValue('item','amount')) > 10000 )
    {
        alert("You cannot add an item with amount greater than 10000.")
        return false;
    }
    return true;
}

```

## Working with Item Groups in a Sublist

NetSuite item groups are stocked and sold as single units, even though they consist of several individual items. Item groups are used to sell vendor-specific objective evidence (VSOE) item group bundles, which can contain both taxable and nontaxable items.

You can use SuiteScript to interact with item groups in the same way you use the UI. Item Group type items are added to transactions as other line items are. In the case of an Item Group item, the item group expands to its member items. Item groups can optionally include start and end lines. The SuiteScript behavior emulates the behavior of how you would add an item group in the UI.

### Example

In this example, an Item Group item is added to a transaction, and the tax code propagates to the members of the group.

```

var rec = nlapiCreateRecord( 'cashesale' );
rec.setFieldValue( 'entity', '76' ); //set the customer
rec.selectNewLineItem( 'item' );
rec.setCurrentLineItemValue( 'item', 'item', '66' ); //item group item
rec.setCurrentLineItemValue( 'item', 'quantity', 1 );
rec.setCurrentLineItemValue( 'item', 'taxcode', -7 ); //set to non-taxable
rec.commitLineItem( 'item' );
var id = nlapiSubmitRecord( rec ); //on submit the item group expands

```

## Working with Sublists in Dynamic Mode and Client SuiteScript

When you copy, create, load, or transform a record in dynamic mode, you are also interacting with all of the record's elements in dynamic mode; this includes a record's sublists.



**Note:** When using client SuiteScript on a sublist, you must also script in a way that emulates the behaviors of the UI. Consequently, an API such as `nlapiSetLineItemValue(type, fldnam, linenum, value)` will generally not be supported in client scripts. Read the rest of this section for more details.



**Note:** If you are unfamiliar with the concept of dynamic scripting, see the help topic [Working with Records in Dynamic Mode](#) for details.

When scripting against a sublist that is in dynamic mode, the following APIs will **NOT** work when adding a line or changing the values of an existing line:

- `nlapiSetLineItemValue(type, fldnam, linenum, value)` - used when scripting in a "current record" context, for example in user event scripts.
- `nlobjRecord.setLineItemValue(group, name, linenum, value)` - used when scripting the `nlobjRecord` object itself, as it exists on the server.

These APIs will not work in dynamic mode or in client SuiteScript because they have no UI correlation. One of the primary components of dynamic and client scripting is that they emulate the behaviors of the UI.

When users interact with sublists in the UI, they first select the sublist they want to work with, then they select the line they want to add or change, and finally they click the Add button to commit the line to the database. When you are scripting a sublist in dynamic mode or in client SuiteScript, calling `nlapiSetLineItemValue(type, fldnam, linenum, value)` does not provide enough context for the script to execute. Instead, you will follow one of these two patterns when adding or changing a line:

### To add a new line:

1. `nlapiSelectNewLineItem(type)` - to specify the sublist you want to work with.
2. `nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)` - to set values on the current line.
3. `nlapiCommitLineItem(type)` - to commit the line to the database.

### Example:

This sample creates a new sales order in dynamic mode, and then adds two new items to the Items sublist.

```
var record = nlapiCreateRecord('salesorder', {recordmode: 'dynamic'});

// add the first item
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'item', 556);
record.setCurrentLineItemValue('item', 'quantity', 2);
record.commitLineItem('item');

// add the second item
record.selectNewLineItem('item');
record.setCurrentLineItemValue('item', 'item', 380);
record.setCurrentLineItemValue('item', 'quantity', '2');
record.setCurrentLineItemValue('item', 'amount', '0.1');
record.commitLineItem('item');
```

## To change values on an existing line:

1. `nlapiSelectLineItem(type, linenum)` - to specify the sublist you want to work with and the existing line you want to change.
2. `nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous)` - to set values on the current line.
3. `nlapiCommitLineItem(type)` - to commit the line to the database.

### Example:

This sample loads a sales order in dynamic mode, and then modifies a line that already exists.

```
var record = nlapiLoadRecord('salesorder', 55, {recordmode: 'dynamic'});

// modify an existing line
record.selectLineItem('item', 1);
record.setCurrentLineItemValue('item', 'item', 556);
record.setCurrentLineItemValue('item', 'quantity', 2);
record.commitLineItem('item');
```

### Example:

This sample loads a sales order in dynamic mode, and then inserts a new item on line one. The item that was previously on line one moves to line two.

```
var record = nlapiLoadRecord('salesorder', 1966, {recordmode: 'dynamic'});
record.insertLineItem('item', 1);
record.setCurrentLineItemValue('item', 'item', 98);
record.commitLineItem('item');
```

## Sublist Errors

You can only set line items that are valid. If you attempt to set a line that does not exist, you will receive an “Invalid Sublist Operation” out-of-bounds error. The exception to this is on Suitelets. Because Suitelets contain only your data, you will not receive a NetSuite error.

### Example

## Working with Sublists in Standard Mode and Client SuiteScript

In standard mode, sublist values are not automatically sourced when you select a record in SuiteScript.

Do the following steps to source values when you select a record:

1. Submit the record with a new placeholder sublist line item.
2. Reload the record.
3. Remove the line item.

4. Set appropriate line item values.
5. Submit the record. After it is submitted, the record is pre-selected on the form for the next new transaction.
6. Reload the record to verify that values are automatically sourced.

## Example

The following example shows how to update the Payment sublist of a Deposit record in standard mode:

```
var rec = nlapiCreateRecord('deposit');

// Set the account.
rec.setFieldValue('account', 123);
// In Standard mode, setFieldValue does not source values in the Payment sublist.

// Insert a placeholder line to enable record submission.
rec.selectNewLineItem('other');
rec.setCurrentLineItemValue('other', 'account', 456);
rec.setCurrentLineItemValue('other', 'amount', 0);
rec.commitLineItem('other');
var id = nlapiSubmitRecord(rec);

// Load the record.
var fin = nlapiLoadRecord('deposit', id);
// The Payment sublist is now populated.

// Remove the placeholder line.
fin.removeLineItem('other', 1);

// Apply values from the Payment sublist.
fin.setLineItemValue('payment','deposit','1','T');

// Submit the finalized record.
var id = nlapiSubmitRecord(fin);
```

# Working with Online Forms

Only the APIs listed in the following table are supported on online forms.



**Important:** These are also the only APIs supported on externally available Suitelets (Suitelets set to Available Without Login on the Script Deployment page). For more information on externally available Suitelets, see [SuiteScript and Externally Available Suitelets](#).

## SuiteScript APIs available on online forms and externally available Suitelets

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>■ nlapiAddDays(d, days)</li> <li>■ nlapiAddMonths(d, months)</li> <li>■ nlapiCancelLineItem(type)</li> <li>■ nlapiDateToString(d, format)</li> <li>■ nlapiDisableField(fldnam, val)</li> <li>■ nlapiDisableLineItemField(type, fldnam, val)</li> <li>■ nlapiEncrypt(s, algorithm, key)</li> <li>■ nlapiEscapeXML(text)</li> <li>■ nlapiFormatCurrency(str)</li> <li>■ nlapiGetCurrentLineItemIndex(type)</li> <li>■ nlapiGetCurrentLineItemText(type, fldnam)</li> <li>■ nlapiGetCurrentLineItemValue(type, fldnam)</li> <li>■ nlapiGetFieldText(fldnam)</li> <li>■ nlapiGetLineItemText(type, fldnam, linenum)</li> <li>■ nlapiIsLineItemChanged(type)</li> <li>■ nlapiRefreshLineItems(type)</li> <li>■ nlapiRemoveLineItemOption(type, fldnam, value)</li> <li>■ nlapiRemoveSelectOption(fldnam, value)</li> <li>■ nlapiSelectLineItem(type, linenum)</li> <li>■ nlapiSelectNewLineItem(type)</li> </ul> | <ul style="list-style-type: none"> <li>■ nlapiGetLineItemCount(type)</li> <li>■ nlapiGetFieldValue(fldnam)</li> <li>■ nlapiSetFieldValue(fldnam, value, firefieldchanged, synchronous)</li> <li>■ nlapiGetLineItemValue(type, fldnam, linenum)</li> <li>■ nlapiSelectNode(node, xpath)</li> <li>■ nlapiSelectNodes(node, xpath)</li> <li>■ nlapiSelectValue(node, xpath)</li> <li>■ nlapiSelectValues(node, path)</li> <li>■ nlapiStringToDate(str, format)</li> <li>■ nlapiStringToXML(text)</li> <li>■ nlapiXMLToString(xml)</li> <li>■ nlapiSetLineItemValue(type, fldnam, linenum, value)</li> <li>■ nlapiInsertLineItem(type, line)</li> <li>■ nlapiRemoveLineItem(type, line)</li> <li>■ nlapiGetRecordType()</li> <li>■ nlapiGetRecordId()</li> <li>■ nlapiGetRole()</li> <li>■ nlapi GetUser()</li> </ul> |
|--|---|



**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.

## Why are only certain APIs supported on online forms?

For security reasons, many SuiteScript APIs are not supported on online forms or externally available (Available Without Login) Suitelets. Online forms and externally available Suitelets are used for generating stateless pages that access or manipulate account information that is not considered to be confidential. Therefore, scripts running on these pages cannot be used to access information on the server because that would require a valid NetSuite session (through user authentication).

Note that server-side SuiteScript execution for such pages (for example, user events and/or Suitelet page generation or backend code) have no such restrictions.



**Note:** `nlapiGetRole()` always returns -31 (the online form user role) when used in this context; `nlapiGetUser()` returns -4 (the return value for a “backdoor” entity).

The APIs listed in the previous section all operate on the current page and will run as expected without a valid NetSuite session. Note that both types of pages (online forms and externally available Suitelets) are hosted on a NetSuite domain called `forms.netsuite.com`. Having a separate domain for online forms and externally available Suitelets prevents secure NetSuite sessions established on `system.netsuite.com` from carrying over to these pages.

NetSuite supports TLS 1.0, 1.1, and 1.2 encryption for `forms.netsuite.com`, `system.netsuite.com`, and other NetSuite domains. Only requests sent using TLS encryption are granted access.

The following figure uses a Suitelet Script Deployment page to show the two domains types. In this case, the Available Without Login preference is selected. When this Suitelet is called, it will be called from the `forms.netsuite.com` domain. So long as only the APIs listed in the table [SuiteScript APIs available on online forms and externally available Suitelets](#) have been used (in addition to any UI Objects), this externally available Suitelet will load and run as intended.

If the Available Without Login preference is not set, the Suitelet will be called from the login domain `system.netsuite.com`



**Note:** Although it is not shown on the Script Deployment record, the internal URL is prepended with `https://system.netsuite.com`.

FIELD	VALUE
SCRIPT	Simple Suitelet Form
STATUS *	Released
TITLE *	Simple Suitelet Form 2
ID	customdeploy2
DEPLOYED	<input checked="" type="checkbox"/>
AVAILABLE WITHOUT LOGIN	<input checked="" type="checkbox"/>
URL	/app/site/hosting/scriptlet.nl?script=115&deploy=2
EXTERNAL URL	https://forms.netsuite.com/app/site/hosting/scriptlet.nl?script=115&deploy=2&compid=563214&h=95b6570f9936f6148918
EVENT TYPE	
LOG LEVEL	Error
EXECUTE AS ROLE	Administrator

# Inline Editing and SuiteScript

The following topics are covered in this section:

- [Inline Editing and SuiteScript Overview](#)
- [Why Inline Edit in SuiteScript?](#)
- [Inline Editing Using nlapiSubmitField](#)
- [Consequences of Using nlapiSubmitField on Non Inline Editable Fields](#)
- [Inline Editing \(xedit\) as a User Event Type](#)
- [What's the Difference Between xedit and edit User Event Types?](#)
- [Inline Editing and nlapiGetNewRecord\(\)](#)
- [Inline Editing and nlapiGetOldRecord\(\)](#)

## Inline Editing and SuiteScript Overview

In the NetSuite UI, inline editing lets you edit fields directly from a record list or from a set of search results. (See the help topic [Using Inline Editing](#) in the NetSuite Help Center for general information on inline editing not related to SuiteScript.)

In SuiteScript, the equivalent of inline editing is changing the value of a field without loading and submitting the entire record the field appears on. This is done using `nlapiSubmitField(type, id, fields, values, doSourcing)`. See [Inline Editing Using nlapiSubmitField](#) for details.

Be aware that in SuiteScript, inline editing and mass updating are considered to be event types that can trigger user event scripts. When users inline edit a field in the UI, or when they perform a mass update, these two event contexts can trigger the execution of a user event script if the script's context type has been set to **xedit**. See [Inline Editing \(xedit\) as a User Event Type](#).



**Important:** When using SuiteScript to inline edit a field on a record, note the following:

- In the UI and in SuiteScript, you can only perform inline editing on **body fields**. You cannot inline edit sublist fields. If you do not know the distinction between body and sublist fields, see [Working with Fields Overview](#) in the NetSuite Help Center.
- In SuiteScript, you cannot inline edit select fields. In other words, you cannot call `nlapiSubmitField` on a select field.
- 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 [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.
- You can use the `nlapiSubmitField` function to inline edit inline-editable body fields on **SuiteScript-supported** records only. Do not use `nlapiSubmitField` (or any other SuiteScript API) on a record that does not officially support SuiteScript. For a list of records that officially support SuiteScript, see [SuiteScript Supported Records](#) in the NetSuite Help Center.
- If you want to perform inline editing through the UI or through SuiteScript, you must first enable the Inline Edit feature in your account. Enable this feature by going to Setup > Company > Enable Features. In the Data Management section, click the Inline Editing check box.

# Why Inline Edit in SuiteScript?

To change values on a record you can either load and submit the entire record, or you can call `nlapiSubmitField(type, id, fields, values, doSourcing)` on a specified body field or fields. Calling the `nlapiSubmitField` function, which is the programmatic equivalent of inline editing, requires less database processing since the entire record object is not being loaded to make a single field update. Note that in the UI and in SuiteScript, not all fields are inline editable.



**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 [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.

## Inline Editing Using `nlapiSubmitField`

The SuiteScript equivalent of inline editing a single field or multiple fields on a record is calling the `nlapiSubmitField(type, id, fields, values, doSourcing)` function. After updating a field's value using `nlapiSubmitField`, you **do not** need to then call `nlapiSubmitRecord` to commit the change to the database.

The following figure shows that the **Phone** field on a customer record is being inline edited in the UI. To save the change, a user needs to click away from the field.

Customers					
VIEW		FILTERS		QUICK SORT	
Leads, prospects, and...ity in the last week		Edit View		(none)	
STYLE	Normal	EDIT	SHOW INACTIVES	EDIT	SEARCH
NEW	EDIT   VIEW	INTERNAL ID	NAME	COMPANY NAME	PHONE
	EDIT   VIEW	226	B2B	BOSICK MEDICAL GROUP	801-529-1340
	Edit   View	226	Boulder Chiropractic Center		406-782-8016
	Edit   View	96	Boulder Cosmetic Dentistry		504-231-2223
	Edit   View	203	Boyd Medical Center		504-315-5400

In SuiteScript, the programmatic equivalent of inline editing the Phone field on customer record 96 is:

```
var updatefield = nlapiSubmitField('customer', '96', 'phone', '504-231-3754');
```

In one call, you can reference a specific record and field, and then set a new value for that field. The entire process consumes only 10 units, which, in many cases, makes updating fields through `nlapiSubmitField` preferable to loading a record, referencing the field on the record, setting a value for the field, and then submitting the entire record to the database. For example, the following script consumes 30 units to accomplish the same thing as the previous inline editing sample:

```
var rec = nlapiLoadRecord('customer', '96'); //10 units
rec.setFieldValue('phone', '504-231-3754');
var id = nlapiSubmitRecord(rec); //20 units
```

Note that with inline editing in SuiteScript you can update multiple fields on a record, and the unit count remains as **10**. In this example, three fields are updated, however, there is still only one call to `nlapiSubmitField`. For example:

```
var fields = new Array();
var values = new Array();
fields[0]='phone';
values[0] = "800-555-1234";
fields[1] = 'url';
values[1] = "www.goodtimeswithsuitescript.com";
fields[2] = 'billpay';
values[2] = "T";
var updatefields = nlapiSubmitField('customer', '149', fields, values);
```

 **Important:** If you are initiating a scheduled script from a user event script, and the user event type is set to xedit, no call to nlapiSubmitField within that scheduled script will actually save the field specified in nlapiSubmitField.

 **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 [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.

## Consequences of Using nlapiSubmitField on Non Inline Editable Fields

In the NetSuite UI, only certain fields are inline editable. These are fields that have no slaving relationship to other fields. When users update a field that is inline editable, only the data for *that* field is updated; there is no cascading effect on other data contained in the record.

Although nlapiSubmitField(...) is the programmatic equivalent of inline editing, it *is* possible to use this API to update fields that are not inline editable in the UI. If a non inline editable field is submitted for update, all the data on the record will be updated appropriately. However, to support this, when a non inline editable field is submitted, the NetSuite backend must load the record, set the field(s), and then submit the record. Completing the “load record, set field, submit record” lifecycle for a record allows all slaving and validation logic on the record to execute.

 **Note:** If an array of fields is submitted using nlapiSubmitField(...), and one field in the array is non inline editable, NetSuite also applies the same solution: the record is loaded in the backend, all fields are set, and the record is submitted.

## Governance Implications

When you use nlapiSubmitField(...) as it is intended to be used (to set one or more fields that are inline editable in the UI), the SuiteScript governance cost is **10 units**.

However, when you use nlapiSubmitField(...) to update fields that are NOT inline editable in the UI, the unit cost for nlapiSubmitField(...) is higher. Your script is charged the units it takes to load and submit a record.

For example, the unit cost of nlapiSubmitField(...) to set a non inline editable field on a transaction is:

1. load the record (nlapiLoadRecord) = 10 units
2. set the field = no units
3. submit the record (nlapiSubmitRecord) = 20 units

Total = 30 units

It is best practice to use nlapiSubmitField(...) as it is intended to be used: to set fields that are inline editable in the UI. To help you know which fields are inline editable, you can refer to the UI.

## Inline Editing (xedit) as a User Event Type

To set a user event script to execute in response to an inline edit field change or a mass update, specify **xedit** as the *type* argument in your script. The **xedit** type can be specified in beforeSubmit or afterSubmit user event scripts.

The following sample shows a user event script that will execute when a user inline edits a record, or the record is updated in a mass update. This script shows how to get all fields that were inline edited on the record or during the mass update.

```
function getUpdatedFields(type)
{
    // if the record is inline edited or mass updated, run the script
    if (type == 'xedit')
    {
        // call nlapiGetNewRecord to get the fields that were inline edited/mass updated
        var fields = nlapiGetNewRecord().getAllFields()

        // loop through the returned fields
        for (var i = 0; i < fields.length; i++)
        {
            if (fields[i] == 'phone')
                nlapiSetFieldValue('phone', nlapiGetFieldValue('phone'))
        }
    }
}
```



**Note:** User event scripts are not executed upon mass updates of child matrix items from their parent items.

## What's the Difference Between xedit and edit User Event Types?

When the user event *type* argument is set to **xedit**, it means that the execution context for the script is inline edit or mass update. In other words, if a user has inline edited a field on a record (or if the record has been part of a mass update), the user event script will execute. In contrast, user event scripts set to execute when the *type* argument is set to **edit** will execute when the record is edited in all other contexts. The script will not execute based on an inline edit or mass update.

## Inline Editing and nlapiGetNewRecord()

In a user event script, if you have set the user event *type* argument to **xedit**, and you are using [nlapiGetNewRecord\(\)](#) to return all the newly updated fields, be aware that only the fields which have been updated through an xedit event (inline edited or mass updated) will be returned. In many cases, this is only one or two fields.

In contrast, if the user event *type* argument is set to **edit**, and you call [nlapiGetNewRecord\(\)](#) in a beforeSubmit, you will get back all the fields on the record.

For **xedit** user events, you should call [nlapiGetNewRecord\(\).getAllFields\(\)](#) to return an array of all the fields being changed in the inline edit, mass update, or [nlapiSubmitField\(\)](#) operation.



**Note:** If you call [getFieldValue\(\)](#) on a field that is not in that array, null is returned.

## Inline Editing and nlapiGetOldRecord()

Although calling [nlapiGetOldRecord\(\)](#) in an inline editing context requires more processing from the NetSuite database (and therefore may add to the user response time), there is less ambiguity when calling this method in an inline editing context than when calling [nlapiGetNewRecord\(\)](#).

The following sample shows how [nlapiGetOldRecord\(\)](#) is used in a user event script that executes in the context of an inline edit or mass update. This script logs all the revised field IDs in the record prior to being committed to the database. If the phone field is modified, the change is reverted.

```
function getUpdatedFields(type)
{
    // if the record is inline edited or mass updated, run this script
    if (type == 'xedit')
    {
        var recOldEmployee = nlapiGetOldRecord();
        var recUpdEmployee = nlapiGetNewRecord();

        // Get all the field IDs in the record
        var lstEmployeeFields = recOldEmployee.getAllFields();

        // Traverse through all the employee fields
        for (var i = 0; i < lstEmployeeFields.length; i++)
        {
            // If the record has a modified phone field, log the original and revised phone numbers
            if (lstEmployeeFields[i] == 'phone')
            {
                nlapiLogExecution('DEBUG', 'Old Phone #', recOldEmployee.getFieldValue('phone'));
            }
            nlapiLogExecution('DEBUG', 'New Phone #', recUpdEmployee.getFieldValue('phone'));

            // Revert the change
            nlapiSetValue('phone', recOldEmployee.getFieldValue('phone'));
        }
    }
}
```

```
    }  
}
```

# Script Types Overview

Script types are organized primarily by where they run (on the client or on the server). They are also organized by the types of tasks you are trying to complete or the data you want to capture.

Use the SuiteScript API to create the following types of scripts.

- [User Event Scripts](#): User Event scripts are triggered when users work with records and data changes in NetSuite as they create, open, update, or save records. User Event scripts are useful for customizing the workflow and association between your NetSuite entry forms. These scripts can also be used for doing additional processing before records are entered or for validating entries based on other data in the system.
- [Suitelets](#): Suitelets enable the creation of dynamic web content. Suitelets can be used to implement custom front and backends. Through API support for scripting forms and lists, these Suitelets can also be used to build NetSuite-looking pages. NetSuite tasklinks can be created to launch a Suitelet. These tasklinks can be used to customize existing centers.
- [RESTlets](#): RESTlets are server-side scripts that can be used to define custom, RESTful integrations to NetSuite. RESTlets follow the principles of the REST architectural style and use HTTP verbs, HTTP headers, HTTP status codes, URLs, and standard data formats. They operate in a request-response model, and an HTTP request to a system-generated URL invokes each RESTlet.
- [Scheduled Scripts](#): Scheduled scripts are executed on-demand in real-time or via a user-configurable schedule. Scheduled scripts are useful for batch processing of records.
- [Client Scripts](#): Client scripts are executed on the client. These scripts can be attached to and run on individual forms, or they can be deployed globally and executed on entity and transaction record types. Global client scripts enable centralized management of scripts that can be applied to an entire record type.
- [Portlet Scripts](#): Portlet scripts are used to create custom dashboard portlets. For example, you can use SuiteScript to create a portlet that is populated on-the-fly with company messages based on data within the system.
- [Mass Update Scripts](#): Mass update scripts allows you to programmatically perform custom mass updates to update fields that are not available through general mass updates. You can also use action scripts to run complex calculations, as defined in your script, across many records.
- [Workflow Action Scripts](#): Workflow action scripts allow you to create custom actions that are defined on a record in a workflow.
- [Bundle Installation Scripts](#): Bundle installation scripts fire triggers that execute as part of bundle installation, update, or uninstall. Trigger execution can occur either before install, after install, before update, after update, or before uninstall. These triggers automatically complete required setup, configuration, and data management tasks for the bundle.

Note that when you create a SuiteScript file, you will need to designate the type of script you want to write. You will do this by going to Setup > Customization > Scripts > New > [type], where **type** is one of the types shown below:

## Select Type

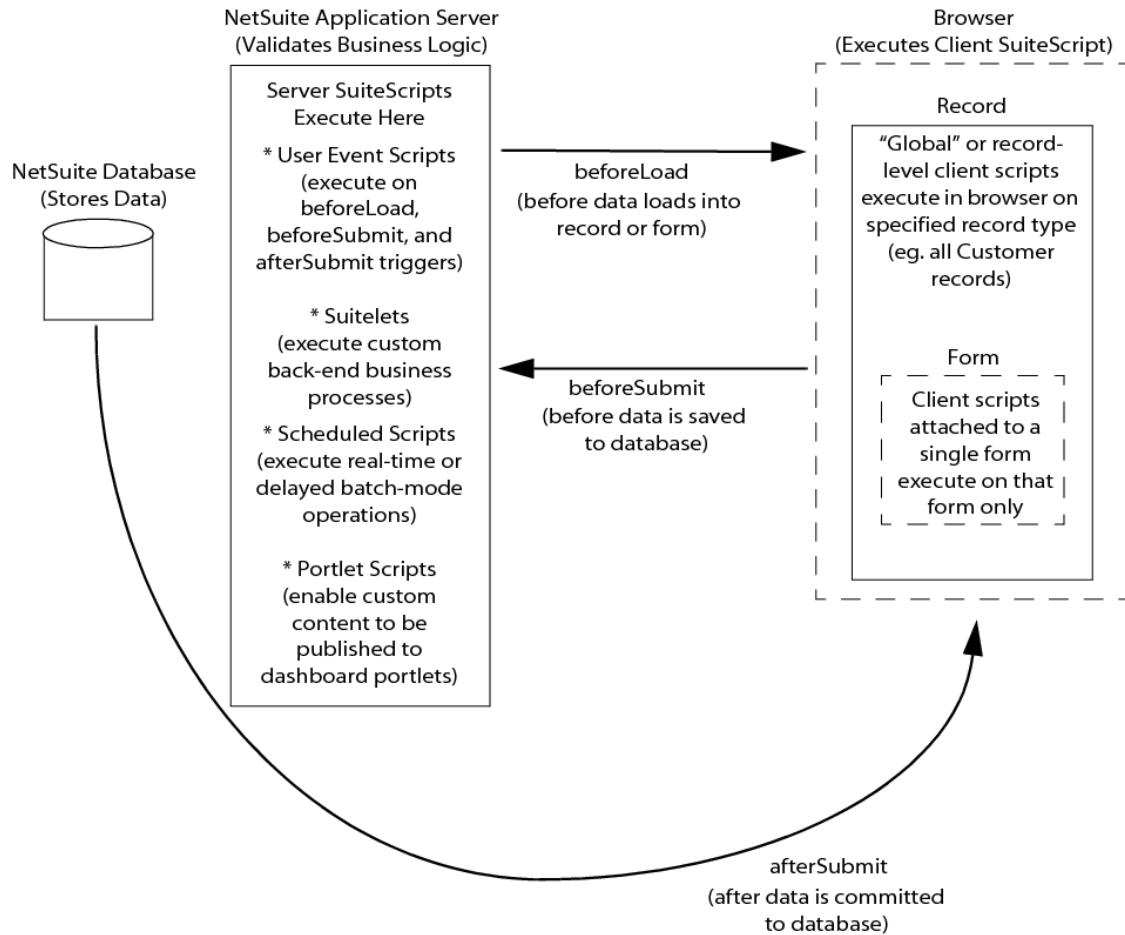
TYPE	DESCRIPTION
Suitelet	Build interactive Web applications by scripting web requests
RESTlet	Build custom RESTful web services
User Event	Define business logic that is triggered when records are created, updated, viewed, or deleted
Scheduled	Schedule complex batch operations or queue them on-demand for execution
Client	Define business logic and perform client-side validation on your forms
Portlet	Publish scriptable portlets to your dashboards and centers
Mass Update	Perform an update to a record as part of a mass update
Workflow Action	Defines a custom action on a record that can be used as part of a workflow
Bundle Installation	Define scripts that run as part of bundle installation or update

To actually run a script in NetSuite, see [Running a Script in NetSuite](#).

# SuiteScript Execution Diagram

When writing your scripts, it is important to understand **where** the scripts will run (client-side or server-side) and **when** the scripts will run (before data loads into a page loads, after an update is made to the data, or after the data has been saved and committed to the database). Understanding the basic concepts of where and when scripts will run will help you understand the SuiteScript API. It will also help you when debugging your code should you encounter problems.

The following diagram provides an overview showing where script types run. For an overview on each script type, see [Script Types Overview](#).



# Client Scripts

The following topics are covered in this section. If you are new to SuiteScript, these topics should be read in order.

- What is Client SuiteScript?
- Client Script Execution
- Client Event Types
- Form-level and Record-level Client Scripts
- Client Script Metering
- Role Restrictions in Client SuiteScript
- How Many Client Events Can I Execute on One Form?
- Error Handling and Debugging Client SuiteScript
- Client Remote Object Scripts
- Running a Client Script in NetSuite
- Client SuiteScript Samples

## What is Client SuiteScript?

Client scripts are SuiteScripts executed in the browser. They can run on most standard records, custom record types, and custom NetSuite pages (for example, Suitelets).



**Note:** To know which standard record types support client SuiteScript, see [SuiteScript Supported Records](#) in the NetSuite Help Center. If a record supports client scripts, an X will appear in the column called “Scriptable in Client SuiteScript”.

Generally, client scripts are used to validate user-entered data and to auto-populate fields or sublists at various form events. Such events can include loading or initializing a form, changing a field, or saving a record. Another use case for client scripts is to source data from an external data source to a field. This is accomplished using the API [nlapiRequestURL\(url, postdata, headers, callback, httpMethod\)](#).

Client scripts are executed by pre-defined event “triggers.” These triggering event types types are discussed in the section [Client Event Types](#). These events include:

- Initializing forms
- Entering or changing a value in a field (before and after it is entered)
- Entering or changing a value in a field that sources another field
- Selecting a line item on a sublist
- Adding a line item (before and after it is entered)
- Saving a form
- Searching for another record
- Loading, saving or deleting a record

After you have created your client script, you can attach your .js script file to the form you are customizing.

If you have created a client script that you want to execute across an entire record type (for example, all Customer records in the system), then you can deploy the script to the specified record type. Client scripts deployed globally affect the behavior of all the records they are deployed to, rather than a specific form on a single record.

## Client Script Execution

Client scripts are executed within a browser. Whether they are client scripts attached to individual forms, or client script deployed globally to an entire record type, all execution occurs in the browser.

Record-level (globally deployed) client scripts are executed **after** any existing form-based client scripts are run, and **before** any user event scripts. This means that record-level client scripts can run on both built-in and custom forms.

Note that there are some scripts that are considered to be client scripts, yet they make calls back to a NetSuite database. In this case you are working with records as “remote objects” on the client.

The following sample is a client script that has been attached to a Sales Order form. (For information on attaching client scripts to forms, see [Step 3: Attach Script to Form](#).) When the user saves the Sales Order, the script gets the value of the **item** field on the Item sublist, then loads a specific Inventory Item record based on the value of the item in the Item sublist. The script then sets a value on the Inventory Item record and submits the record. Although there is backend activity being executed in this script, the script's initial execution is based on the `saveRecord` client event trigger (see [Client Event Types](#)), therefore it is still considered to be a client script.

```
// Client side script on sales order, on save
// Load the 1st item and mark it inactive
function onSave()
{
    var id = nlapiGetLineItemValue('item', 'item', 1);
    var record = nlapiLoadRecord('inventoryitem', id);
    record.setFieldValue('isinactive', 'T');
    nlapiSubmitRecord(record);

    return true;
}
```

## Client Event Types

In NetSuite, client scripts can be executed on 10 different client-side events. These client events can occur when a user loads a NetSuite form into the browser, or when a user selects a field or a field is updated. Field updates can occur when a user updates a field, or when a field is auto-updated through a sourcing relationship with another field. A client event can also occur when a user clicks the Submit or Save button on a NetSuite page.

The following table describes each client event type and the actions associated with the event. Note that the functions that are called on each event type do not have to be written as `pageInit()` or `fieldChanged()`, and so on. However, when writing your client script, it is best practice to indicate the

event type in the function name, for example: `pageInit_alertSalesRep()`, or `validateField_department()`, or `saveRecordCustomer()`.

Client Event Type (and sample function name)	Parameters	Returns	Description
pageInit	<code>type</code> : the mode in which the record is being accessed. These modes can be set to: <ul style="list-style-type: none"><li>■ <code>create</code></li><li>■ <code>copy</code></li><li>■ <code>edit</code></li></ul>		<p>This client event occurs when the page completes loading or when the form is reset. This function is automatically passed the <code>type</code> argument from the system.</p> <p>This is similar to an <code>onLoad</code> JavaScript client-side event.</p> <p>See <a href="#">Page Init Sample</a>.</p>
saveRecord		Boolean	<p>This client event occurs when the submit button is pressed but prior to the form being submitted. You should always return either <code>true</code> or <code>false</code> from a <code>saveRecord</code> event. A return value of <code>false</code> suppresses submission of the form.</p> <p>See <a href="#">Save Record Sample</a>.</p>
validateField	<code>type</code> : the sublist internal ID <code>name</code> : the field internal ID <code>linenum</code> : line number if this is a sublist. Line numbers start at 1, not 0.	Boolean	<p>This client event occurs whenever a field is about to be changed by the user or by a client side call. Returning <code>false</code> from this function prevents the field's value from changing.</p> <p>This function is automatically passed up to three arguments by the system: <code>type</code>, <code>name</code>, <code>linenum</code>.</p> <p>This event type is similar to an <code>onBlur</code> JavaScript client-side event.</p> <p>In NetSuite, validateField events execute on fields added in beforeLoad user event scripts.</p> <p>Note: This event type does NOT apply to drop-down select or check box fields.</p> <p>See <a href="#">Validate Field Sample</a>.</p>
fieldChanged	<code>type</code> : the sublist internal ID <code>name</code> : the field internal ID <code>linenum</code> : line number if this is a sublist. Line numbers start at 1, not 0.		<p>This client event occurs whenever a field is changed by the user or by a client side call. This event can also occur directly through beforeLoad user event scripts.</p> <p>This client event does not occur when field information is changed or appears in the page URL. Use the <code>pageInit</code> event to handle URLs that may contain updated field values.</p> <p>This function is automatically passed up to three arguments by the system: <code>type</code>, <code>name</code>, <code>linenum</code>.</p> <p>This event type is similar to an <code>onChange</code> JavaScript client-side event.</p> <p>See <a href="#">Field Changed Sample</a>.</p>

Client Event Type (and sample function name)	Parameters	Returns	Description
postSourcing	<i>type</i> : the sublist internal ID <i>name</i> : the field internal ID		This client event occurs following a field change after all of the field's child field values are sourced from the server. Enables fieldChange style functionality to occur after all dependent field values have been set. This function is automatically passed up to two arguments from the system: <b>type, name</b> . See <a href="#">Post Sourcing Sample</a> .
lineInit	<i>type</i> : the sublist internal ID		This client event occurs when an existing line is selected. It can be thought of as the pagelinit function for sublist line items (inlineeditor and editor sublists only). This function is automatically passed the <b>type</b> argument from the system.
validateLine	<i>type</i> : the sublist internal ID	Boolean	This client event occurs prior to a line being added to a sublist (inlineeditor or editor sublists only). It can be thought of as the saveRecord equivalent for sublist line items (inlineeditor and editor). Returns false to reject the operation. References to fields should be done using nlapiGetCurrent*** functions. This function is automatically passed the <b>type</b> argument from the system.
recalc	<i>type</i> : the sublist internal ID		This event occurs after a sublist change, but only if the sublist change causes the total to change. This event is designed to be used for updating a global total, not for manipulating the current line item value. Do not call any getCurrentLineItem or setCurrentLineItem functions for this event; they will not work! Instead use the validateLine event. Notes: <ul style="list-style-type: none"><li>■ Recalc functions will not execute when the <b>Add Multiple</b> button is used to add multiple line items.</li><li>■ Scripts that execute on recalc events do not need to include a call to <a href="#">nlapiCommitLineItem(type)</a> since recalculation occurs automatically. Calling nlapiCommitLineItem will end up calculating the line twice.</li></ul>
validateInsert	<i>type</i> : the sublist internal ID	Boolean	The validateInsert event occurs when you insert a line into an edit sublist. For

Client Event Type (and sample function name)	Parameters	Returns	Description
			<p>information on the edit sublist type, see <a href="#">Editor Sublists</a> in the NetSuite Help Center.</p> <p>The UI equivalent of this event is when a user selects an existing line in a sublist and then clicks the Insert button. In SuiteScript, the equivalent action is calling <code>nlobjRecord.insertLineItem(...)</code>. Note that returning false on a validateInsert blocks the insert.</p>
validateDelete	<code>type</code> : the sublist internal ID	Boolean	<p>The validateDelete event occurs when you try to remove an existing line from an edit sublist. Returning false blocks the removal.</p> <p>For information on the edit sublist type, see <a href="#">Editor Sublists</a> in the NetSuite Help Center.</p>

\*The ValidateField and FieldChanged scripts require a null line item for body fields.

## Form-level and Record-level Client Scripts

Form-based client scripts run against specific fields and forms. Record-level client scripts, similar to user event scripts, are deployed globally and run against an entire record type. For example, record-level client scripts can be deployed to run against all Invoice records or all Customer records in the system.

Record-level client scripts run independent of any client scripts already attached to a specific form on the record. Record-level client scripts can also be used on forms and lists that have been generated through [UI Objects](#) during [Suitelets](#) development. Form-based client scripts cannot be used by Suitelets.

Additionally, record-level clients scripts allow you to set audience definitions on the Script Deployment page. Defining an audience for the script deployment allows you to specify which members of your company, which departments, and which roles will be able to see the record-level customizations that have been implemented.

To deploy record-level client scripts into your account, you must follow the deployment model used for Suitelet, user event, scheduled, and portlet scripts. (See [Running Scripts in NetSuite Overview](#) to learn how to run a record-level script in NetSuite.)

Form-level client scripts, however, require only that the client script is attached to the individual form it is running against. For details on attaching client scripts to forms, see [Step 3: Attach Script to Form](#).



**Note:** You can deploy up to 10 record-level client script to any record types that are already supported in the existing form-based model. For information on records supported in client scripting, see [SuiteScript Supported Records](#) in the NetSuite Help Center.

## Change to Permission Required for Attaching Form-Level Scripts

Prior to Version 2016 Release 1, users who had the Custom Address Form, Custom Entry Forms, or Custom Transaction Forms permission, but who did not have the SuiteScript permission, could edit the Custom Code tab of a form record to attach a script to the form. As of this release, these users can no longer access the Custom Code tab of custom forms.

Now users must have at least the Edit level of the SuiteScript permission to attach a script to a custom form by editing the Custom Code tab of the form record. For users with the Edit or Full level of the SuiteScript permission, the Custom Code tab is displayed and is fully editable. Users with the View or Create level of the SuiteScript permission can see the Custom Code tab, but cannot edit it. For users who do not have SuiteScript permission, the Custom Code tab is not visible.

## Client Script Metering

Client scripts are metered or governed on a per-script basis. For example, if an account has one **form-level** client script attached to a form, and one **record-level** client script deployed to the record (which contains the form), **each** client script can total 1000 units. Units are not shared among the client scripts that are associated with a form or record.



**Note:** For information on script metering and the SuiteScript governance model, see [SuiteScript Governance](#).

## Role Restrictions in Client SuiteScript

Running a client script to access a record will result in an error if the role used does not have permission to view/edit that record. Client SuiteScript respects the role permissions specified in the user's NetSuite account.

### Example

The following is a client script, which you can attach to a custom sales order form and set to execute on the field change client event.

```
function email(){
    var salesRep = nlapiGetFieldValue('salesrep');
    var salesRepEmail = nlapiLookupField('employee', salesRep, 'email');
    alert(salesRepEmail);
}
```

If you are logged in as admin, when you load the sales order with this form, and then select the Sales Rep field, you will receive the alert. However, if you log in using a non-admin role (such as Sales



Manager), or a role that does not have permission to view/edit Employee records, you will receive an error when you select the Sales Rep field.

To work around this issue, as the script developer you must consider the types of users who may be using your custom form and running the script. Consider which record types they do and do not have access to. If it is vital that all who run the script have access to the records in the script, you may need to redefine the permissions of the users (if your role is as an admin). Or you may need to rewrite your script so that it references only those record types that all users have access to.

Another consideration is to write the script as a server-side user event script and set the “Execute As Admin” preference on the script’s Script Deployment page. Note that in the sample script above, you would not be able to run the script as a user event script and throw an alert, as alerts are a function of client scripts only. However, you could rewrite the script so that it emails the user the sales rep’s email address (instead of throwing an alert).



**Note:** For information on user event scripts, see [User Event Scripts](#). For information on executing scripts as admin, see [Executing Scripts Using a Specific Role](#).

## How Many Client Events Can I Execute on One Form?

Client scripts have a limit of 10 client events per form. The following figure shows the Custom Code tab on a Custom Entry Form for a Customer record. If you choose, you can run a script that contains client event functions for all 10 available event types. This figure shows only five client event functions are specified within this script, but all 10 in one script file is supported.

For information on client event functions, see [Client Event Types](#).

Event Type	Function Name
SCRIPT FILE	field_changed.js
PAGE INIT FUNCTION	samplePageInit
SAVE RECORD FUNCTION	sampleSaveRec
VALIDATE FIELD FUNCTION	sampleValidateField
FIELD CHANGED FUNCTION	sampleFieldChanges
POST SOURCING FUNCTION	
LINE INIT FUNCTION	
VALIDATE LINE FUNCTION	
VALIDATE INSERT FUNCTION	sampleValidateInsert
VALIDATE DELETE FUNCTION	
RECALC FUNCTION	



**Note:** For information on attaching a client script to a form and running the script in your account, see Step 3: Attach Script to Form.

## Error Handling and Debugging Client SuiteScript



**Important:** You cannot debug form or record-level client scripts in the SuiteScript Debugger. To debug client scripts, NetSuite recommends using either the Firebug debugger, which integrates with Firefox, or the Microsoft Script Debugger, which integrates with Internet Explorer. For instructions on working with either of these debuggers, please see the documentation provided with each product.

Regarding error handling in client SuiteScript, NetSuite catches and throws alerts for all unhandled SuiteScript errors that occur in both form- and record-level client scripts.

Note that alerts provide the scriptId of the Script record. This is information that will help NetSuite administrators locate the specific SuiteScript file that is throwing the error.

Also note that like other script types, the Script record page for **record-level** client scripts includes an Unhandled Errors subtab. NetSuite administrators can use this tab to define which individual(s) will be notified if script errors occur. For additional information on the Unhandled Errors subtab, see [Steps for Creating a Script Record](#).

Additionally, the Script Deployment page for **record-level** client scripts includes an Execution Log subtab, on which all script errors are logged.

## Client Remote Object Scripts

A client remote object script is a client-side SuiteScript that makes a call to the NetSuite server to create, load, copy, or transform a record object.

The following is an example of a client remote object script. On the saveRecord client event, the script executes the nlapiCreateRecord(...) function to create a new estimate record. This script creates the new Estimate, sets values on the new record, and then submits the record, all from within a script that is executed on the client.

### Example

```
function onSave()
{
// access the NetSuite server to instantiate a new Estimate
var rec = nlapiCreateRecord('estimate'); rec.setFieldValue('entity','846');
rec.insertLineItem('item',1);
rec.setLineItemValue('item','item', 1, '30');
rec.setLineItemValue('item','quantity', 1, '500');

var id = nlapiSubmitRecord(rec, true);

return true;
}
```



**Important:** You cannot use client remote object scripts to access a remote object in dynamic mode. See the help topic [Client Scripting and Dynamic Mode](#) for details.

## Running a Client Script in NetSuite

To run a client script in NetSuite, you must:

1. Create a JavaScript file for your client script.
2. Load the file into NetSuite.
3. Attach your script file to a custom form (if you have written a form-level client script).
4. Create a Script record (if you have written a record-level client script).
5. Define all runtime options on the Script Deployment page (if you have written a record-level client script).

If you are new to SuiteScript and need information on each of these steps, see [Running Scripts in NetSuite Overview](#).

## Client SuiteScript Samples

The following samples are covered in this section. They illustrate how client event functions are used to interact with the form.

- [Writing Your First Client Script](#)
- [Page Init Sample](#)
- [Save Record Sample](#)
- [Post Sourcing Sample](#)
- [Validate Field Sample](#)
- [Field Changed Sample](#)

## Writing Your First Client Script

A great way to get started with client scripts is to deploy a script that has a function on every exposed event. Consider the following client script:

```
function myPageInit(type)
{
    alert ('myPageInit' );
    alert ('type=' + type);
}

function mySaveRecord()
{
    alert ('mySaveRecord' );
    returntrue ;
}

function myValidateField(type, name, linenum)
{
    if (name === 'custentity_my_custom_field' )
```

```
{  
    alert ('myValidateField' );  
    alert ('type=' + type);  
    alert ('name=' + name );  
    alert ('linenum=' + linenum);  
}  
returntrue ;  
}  
  
function myFieldChanged(type, name, linenum)  
{  
    alert ('myFieldChanged' );  
    alert ('type=' + type);  
    alert ('name=' + name );  
    alert ('linenum=' + linenum);  
}  
  
function myPostSourcing(type, name )  
{  
    alert ('myPostSourcing' );  
    alert ('type=' + type);  
    alert ('name=' + name );  
}  
  
function myLineInit(type)  
{  
    alert ('myLineInit' );  
    alert ('type=' + type);  
}  
  
function myValidateLine(type)  
{  
    alert ('myValidateLine' );  
    alert ('type=' + type);  
}  
  
function myValidateInsert(type)  
{  
    alert ('myValidateInsert' );  
    alert ('type=' + type);  
}  
  
function myValidateDelete(type)  
{  
    alert ('myValidateDelete' );  
    alert ('type=' + type);  
}  
  
function myRecalc(type)  
{  
    alert ('myRecalc' );  
    alert ('type=' + type);  
}
```

This sample displays all the available arguments for every script-triggered event. Notice that some functions return a Boolean while some do not. Also note that some functions have **linenum** as one of the arguments. Sublist functions do not have a **linenum** argument because the event is confined to the specific line that triggered it.

The function `myValidateField` has an additional `if` block to check whether the event was invoked by a custom field with the id `custentity_my_custom_field`. This ensures the logic is executed only under the correct circumstances.

**Note:** It is important to check the argument values to branch execution logic. This improves performance and avoids logic executed indiscriminately.

To obtain a better understanding on when these client script events are triggered and what the arguments contain, upload the JavaScript file to the SuiteScript folder in the NetSuite file cabinet, and deploy the script by specifying the functions in a script record.

The screenshot shows the 'Script' setup page in NetSuite. At the top, there are buttons for 'Save & New' (highlighted in blue), 'Cancel', and 'Reset'. The 'TYPE' is set to 'Client'. The 'NAME \*' field contains 'My client script'. The 'ID' field contains '\_wlf\_my\_first\_client\_script'. On the right side, there are fields for 'DESCRIPTION' (empty), 'OWNER' (Wolfe, K), and an 'INACTIVE' checkbox (unchecked). Below these fields, there is a section titled 'SCRIPT FILE \*' with a dropdown menu showing '<Type then tab>'. The main area lists various client-side functions: 'PAGE INIT FUNCTION' (myPageInit), 'SAVE RECORD FUNCTION' (mySaveRecord), 'VALIDATE FIELD FUNCTION' (myValidateField), 'FIELD CHANGED FUNCTION' (myFieldChanged), 'POST SOURCING FUNCTION' (myPostSourcing), 'LINE INIT FUNCTION' (myLineInit), 'VALIDATE LINE FUNCTION' (myValidateLine), 'VALIDATE INSERT FUNCTION' (myValidateInsert), 'VALIDATE DELETE FUNCTION' (myValidateDelete), and 'RECALC FUNCTION' (myRecalc). Each function name is preceded by a small blue link icon.



**Note:** When saved the Script record is saved, the system will automatically prefix the script ID with **customscript**. In the figure above, the final unique ID for this client script will be **customscript\_wlf\_my\_client\_script**. This custom script record identifier can be passed as a value to the **scriptId** parameter that is included in several SuiteScript APIs.

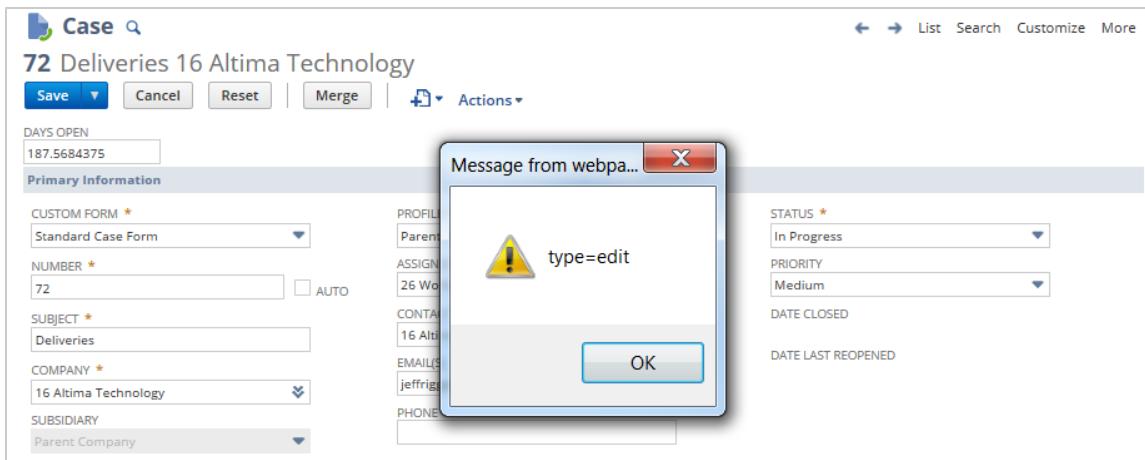
The previous screenshot demonstrates the definition of a record-level client script. This script is deployed globally to any records specified in the Deployments tab (see figure). In this case, this script will be deployed to all **Case** records in the system.

The screenshot shows the 'Script' setup screen. The 'TYPE' is set to 'Client'. The 'NAME' is 'My client script'. The 'ID' is 'customscript\_wlf\_my\_first\_client\_script'. The 'DESCRIPTION' and 'OWNER' fields are empty. A checked checkbox indicates 'INACTIVE'. The 'Deployment' tab is selected, showing the deployment details: 'APPLIES TO' is 'Case', 'ID' is 'customdeploy1', 'DEPLOYED' is 'Yes', 'STATUS' is 'Testing', and 'EVENT TYPE' is empty. There is also an 'Add Multiple' button.

When a person opens any Case record, the following alerts are thrown right way on the pagelinit (page load) trigger:

The screenshot shows a 'Case' record page for number 72. The primary information includes fields like CUSTOM FORM, NUMBER, SUBJECT, COMPANY, and SUBSIDIARY. A modal alert window titled 'Message from webpa...' is displayed in the center. The alert contains a yellow warning icon, the text 'myPageInit', and an 'OK' button. The background of the page shows other fields like PROFILE, ASSIGN, CONTACT, EMAIL, and PHONE.

When you click **OK** to begin editing the record, the type=edit alert is thrown.



## Page Init Sample

The Page Init function is called when the form is first loaded. Some of the functions that can be performed on Pagelnit include the following:

- Populate field defaults
- Disable or enable fields
- Change field availability or values depending on the data available for the record
- Add flags to set initial values of fields
- Provide alerts where the data being loaded is inconsistent or corrupt
- Retrieve user login information and change field availability or values accordingly
- Validate that fields required for your custom code (but not necessarily required for the form) exist

## Examples

### Set Default Field Values for a Field

```
function pagelnit()
{
    // if fieldA is either NULL or equal to "valueA"
    if ((nlapiGetFieldValue('fieldA').length === 0) || (nlapiGetFieldText('fieldA') === "valueA"))
    {
        // then set fieldA to valueB
        nlapiSetFieldText('fieldA', nlapiGetFieldText('valueB'));
    }
}
```

### Disable a Field

```
function pagelnit()
{
    //On init, disable two optional Other fields: fieldA and fieldB.
    nlapiDisableField('custrecord_other_fieldA', true);
```

```

    nlapiDisableField('custrecord_other_fieldB', true);
}

```

## Display User Profile Information

```

function pageInit()
{
    //On page init display the currently logged in User's profile information.

    // Set variables
    var userName = nlapiGetUser();      // entity id of the current user
    var userRole = nlapiGetRole();      // id of the current user's role
    var userDept = nlapiGetDepartment(); // id of the current user's department
    var userLoc = nlapiGetLocation();   // id of the current user's location

    // Display information
    alert("Current User Information" + "\n\n" +
        "Name: " + userName + "\n" +
        "Role: " + userRole + "\n" +
        "Dept: " + userDept + "\n" +
        "Loc: " + userLoc
    );
}

```

## Save Record Sample

The Save Record function is called when the user requests the form to be saved. This function returns false to reject the operation. Use the Save Record function to provide alerts to the user before committing the data. If it is necessary for the user to make changes before committing the data, return false — otherwise display the alert, return true and allow the user to commit the data.

You can also use the Save Record function to:

- Enable fields that were disabled with other functions
- Redirect the user to a specified URL

## Examples

### Requesting Additional Information

```

function saveRecord()
{
    // Check to see that fieldA is populated. If not, block the save and warn with a popup.

    if (String(nlapiGetFieldValue('fieldA')).length === 0)
    {
        alert("Please provide a value for fieldA");
        return false;
    }
    alert("Are you sure you want to Save the record?");
    return true;
}

```

## Redirect the User to Another Location

```
function saveRecord()
{
    window.open('https://system.netsuite.com/[url string]');void(0)
    return true;
}
```

## Post Sourcing Sample

(Transaction Forms Only)

The Post Sourcing function is called when a field is modified that sources information from another field. Event handlers for this function behave similar to event handlers for the Change Field function except that the function is called only after all sourcing is completed — it waits for any slaved or cascaded field changes to complete before calling the user defined function. Therefore, the event handler is not triggered by field changes for a field that does not have any slaved fields.

If there is at least one field sourced from a drop down (either a built-in sourcing or one created through customization) the post sourcing event is fired. Therefore, if you need to do something based on sourced values, you should do it in Post Sourcing rather than from Field Changed.

## Example

### On Sales order – Post sourcing

```
// Wait for all sourcing to complete from item field, get the rate field. If rate < 10, set it
to 20.
// Execute this post sourcing function
function postSourcing(type, name)
{
    // Execute this code when all the fields from item are sourced on the sales order.

    if(type === 'item' && name === 'item')
    {
        // After all the fields from item are sourced
        var rate = nlapiGetCurrentLineItemValue('item', 'rate');
        var line = nlapiGetCurrentLineItemIndex(type);

        if(rate < 10)
        {
            nlapiSetCurrentLineItemValue('item', 'rate', 20);
        }
    }
}
```

## Validate Field Sample

The ValidateField function is called whenever the user **changes** the value of a field. This function returns false to reject the value.





**Note:** This event type does not apply to drop-down or check box fields.

Use the Validate Field function to validate field lengths, restrict field entries to a predefined format, restrict submitted values to a specified range, validate the submission against entries made in an associated field,

## Examples

### Validate Field Lengths

```
function ValidateField(type, name)
{
    // if fieldA is not at least 6 characters, fail validation
    if (name === 'fieldA')
    {
        var fieldALength = String(nlapiGetFieldValue('fieldA')).length;

        if (fieldALength < 6)
        {
            alert("FieldA must be at least 6 characters.");
            return false;
        }
    }
    // Always return true at this level, to continue validating other fields
    return true;
}
```

### Validate Field is Uppercase

This sample uses a validate field function that ensure the value in the field with ID custrecord\_mustbe\_uppercase is always set to uppercase.

```
function validateFieldForceUppercase(type, name)
{
    if (name === 'custrecord_mustbe_uppercase')
    {
        //obtain the upper case value
        var upperCase = nlapiGetFieldValue('custrecord_mustbe_uppercase').toUpperCase();

        //make sure it hasn't been set
        if (upperCase !== nlapiGetFieldValue('custrecord_mustbe_uppercase'))
        {
            nlapiSetFieldValue('custrecord_mustbe_uppercase', upperCase, false);
        }

        return true ;
    }
}
```

Since this function is invoked every time there is an attempt to move the focus away from a field, the first if block ensures the uppercase logic is executed only for the correct field. Since using the API nlapiSetFieldValue would also trigger events, the second if block is put in place to ensure the code will

not get into an infinite loop. The final return true statement ensures the focus can be successfully taken away from the field.

## Field Changed Sample

The Field Changed function is called when a new value for a field is **accepted**. Use the Field Changed function to provide the user with additional information based on user input, disable or enable fields based on user input.

### Examples

#### Requesting Additional Information

```
function FieldChanged(type, name)
{
    // Prompt for additional information, based on values already selected.
    if ((name === 'fieldA') && (nlapiGetFieldText('fieldA') === "Other"))
    {
        alert("Please provide additional information about fieldA
              in the text field below.");
    }
}
```

# User Event Scripts

The following topics are covered in this section. If you are new to user event scripts, these topics should be read in order:

- What Are User Event Scripts?
- User Event Script Execution
- Setting the User Event type Argument
- User Event Script Execution Types
- How Many User Events Can I Have on One Record?
- Running a User Event Script in NetSuite
- User Event Script Samples

## What Are User Event Scripts?

User event scripts are executed on the NetSuite server. They are executed when users perform certain actions on records, such as create, load, update, copy, delete, or submit. Most standard NetSuite records and custom record types support user event scripts.

 **Important:** User event scripts cannot be executed by other user event scripts or by workflows with a **Context of User Event Script**. In other words, you cannot chain user event scripts. You can, however, execute a user event script from a call within a NetSuite scheduled script, a portlet script, or a Suitelet.

With user event scripts you can do such things as:

- Implement custom validation on records
- Enforce user-defined data integrity and business rules
- Perform user-defined permission checking and record restrictions
- Implement real-time data synchronization
- Define custom workflows (redirection and follow-up actions)
- Implement custom form customizations

 **Note:** To know which standard record types support user event scripts, see [SuiteScript Supported Records](#) in the NetSuite Help Center. If a record supports user event scripts, an X will appear in the column called "Scriptable in Server SuiteScript".

## Which User Event Types are Available in SuiteScript?

The user events types that are available in scripting are:

- Before Load – event occurs when a read operation on a record takes place. A Before Load event occurs when a user clicks Edit or View on an existing record, or clicks New to create a new record.
- Before Submit – event occurs when a record is submitted, but before the changes are committed to the database. A Before Submit event occurs when a user clicks Save (or Submit) on a record.
- After Submit – event occurs **after** the changes to the record are committed to the database. An After Submit event occurs after a user has clicked Save (or Submit) on a record.

See [User Event Script Execution Types](#) or specific details on these event types.

## User Event Script Execution

User event scripts are executed based on operation types defined as: `beforeLoad`, `beforeSubmit`, and `afterSubmit`. See the following sections for details:

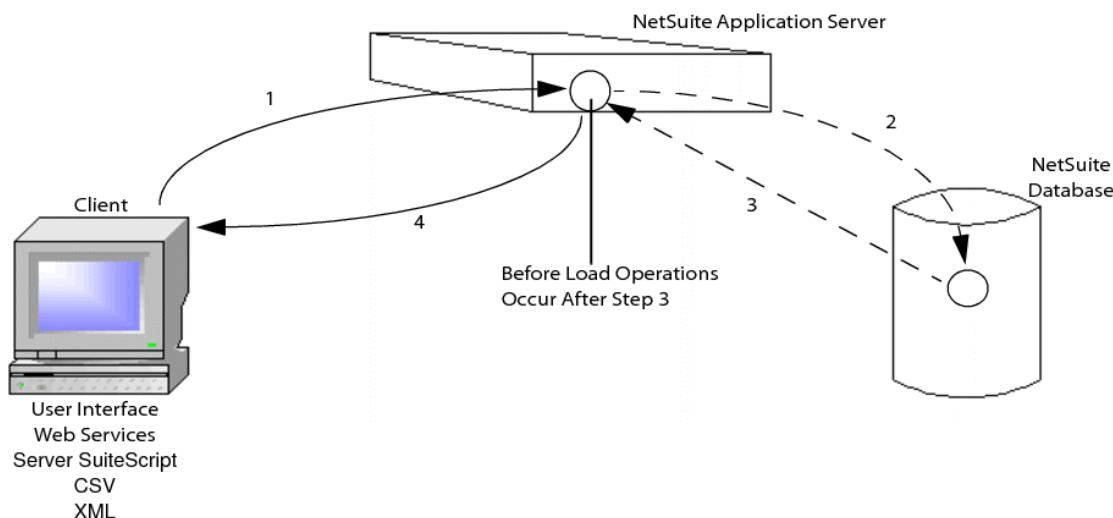
- [User Event beforeLoad Operations](#)
- [User Event beforeSubmit and afterSubmit Operations](#)

For information on the arguments each user event function takes, see [Setting the User Event type Argument](#).

### User Event beforeLoad Operations

The following steps and diagram provide an overview of what occurs during a `beforeLoad` operation:

1. The client sends a read operation request for record data. (The client request can come from the user interface, web services, server-side SuiteScript calls, CSV imports, or XML.)
2. Upon receiving the request, the application server performs basic permission checks on the client.
3. The database loads the requested information into the application server for processing. This is where the `beforeLoad` operation occurs – before the requested data is returned to the client.
4. The client receives the now validated/processed `beforeLoad` data.



**Note:** Standard records cannot be sourced during a `beforeLoad` operation. Use the `pageInit` client script for this purpose. See [Client Scripts](#).

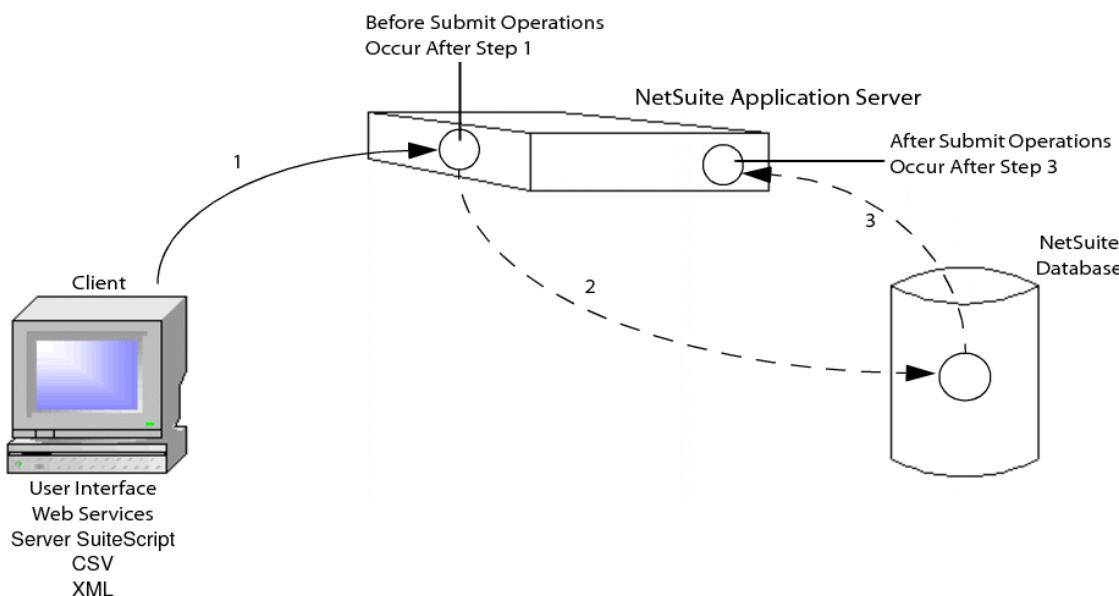
### User Event beforeSubmit and afterSubmit Operations

The following steps and diagram provide an overview of what occurs on submit (`beforeSubmit` and `afterSubmit`) operations:

1. The client performs a write operation by submitting data to the application server. (The client request can come from the user interface, web services, server-side SuiteScript calls, CSV imports, or XML.) The application server:
  - a. performs basic permission checks on the client
  - b. processes the submitted data and performs specified validation checks during a `beforeSubmit` operation

The submitted data has **NOT** yet been committed to the database.
2. After data has been validated, it is committed to the database.
3. If this (newly committed) data is then called by an `afterSubmit` operation, the data is taken from the database and is sent to the application server for additional processing. Examples of `afterSubmit` operations on data that are already committed to the database include, but are not limited to:
  - a. sending email notifications (regarding the data that was committed to the database)
  - b. creating child records (based on the data that was committed to the database)
  - c. assigning tasks to employees (based on data that was committed to the database)

**Note:** Asynchronous `afterSubmit` user events are only supported during webstore checkout.



## Setting the User Event type Argument

For User Event scripts, you can associate a script execution context with the `type` argument of the script's function. For example, if you have a script associated with a `beforeLoad` operation for a specific record type, and you would like to cause an action only when the record is initially created, specify `create` as the script execution type.

**Important:** The `type` argument is an **auto-generated** argument passed by the system. You can NOT set this as a parameter for a specific deployment like other function arguments.

```
//Define the User Event function for a beforeLoad operation.
```

```

function beforeLoadSalesOrder( type )
{
    var newRecord = nlapiGetNewRecord();
    var cutoffRate = custscript_maximumdiscountlevel;
    var discountRate = newRecord.getFieldValue('discountrate');

    //Define the value of the type argument.
    if (type == 'create' && discountRate != null && discountRate.length > 0
        && cutoffRate != null && cutoffRate.length > 0)
    {
        discountRate = Math.abs( parseFloat( discountRate ) );
        ...remainder of code...
    }
}

```

Event type arguments vary depending on whether the event will occur on beforeLoad, beforeSubmit, or afterSubmit operations. The following table lists the script execution event types you can use with each operation.

**i Note:** When deploying user event scripts in NetSuite, you can also define a script execution event type using the Event Type drop-down list on the Script Deployment page. Be aware that the event type you choose from the drop-down list will override the type(s) specified in the actual script. For details, see [Setting Script Execution Event Type from the UI](#). For general information on defining other deployment parameters for User Event scripts, see [Steps for Defining a Script Deployment](#).

All of the events have the common type argument which indicates the type of operation that invoked the event. This argument allows the script code to branch out to different logic depending on the operation type. For example, a script with “deltree” logic that deletes a record and all of its child records should only be invoked when type equals to “delete”. It is very important that user event scripts check the value of the type argument to avoid indiscriminate execution.

The following sample demonstrates how to check the value of the type argument for each event. Event types include beforeLoad, beforeSubmit, afterSubmit. (See [User Event Script Execution Types](#) for more details.)

```

function myBeforeLoadUE(type)
{
    if (type == 'create')
    {
        nlapiLogExecution('DEBUG', 'type argument', 'type is create');
    }

    if (type == 'view')
    {
        nlapiLogExecution('DEBUG', 'type argument', 'type is view');
    }

    if (type == 'edit')
    {
        nlapiLogExecution('DEBUG', 'type argument', 'type is edit');
    }
}

function myBeforeSubmitUE(type)

```

```
{
  if (type == 'create')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is create');
  }

  if (type == 'delete')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is delete');
  }

  if (type == 'edit')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is edit');
  }

  if (type == 'cancel')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is cancel');
  }
}

function myAfterSubmitUE(type)
{
  if (type == 'create')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is create');
  }

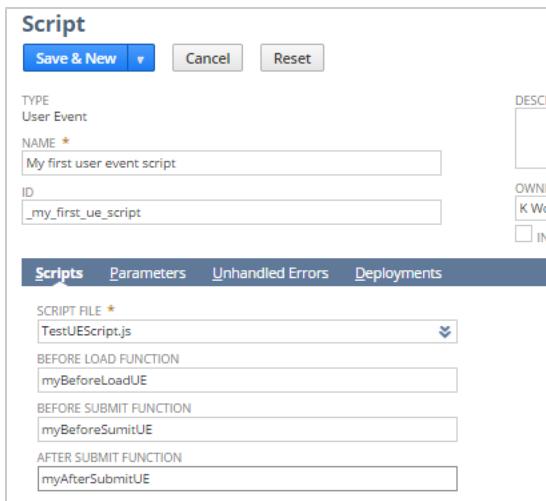
  if (type == 'delete')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is delete');
  }

  if (type == 'edit')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is edit');
  }

  if (type == 'approve')
  {
    nlapiLogExecution('DEBUG', 'type argument', 'type is approve');
  }
}
```

**i Note:** Logging done with nlapiLogExecution may be classified into 4 types: DEBUG, AUDIT, ERROR, and EMERGENCY. The source code should correctly set the logging type. Log type filtering may be set during runtime to give concise and useful logged information.

After uploading the source code file to the SuiteScript folder in the File Cabinet, a user event script record is defined by going to Set Up > Customization > Scripts > New > User Event. The following shows the script record definition page.



## User Event Script Execution Types

The following table lists all the execution context types that are supported in each user event type:

Operation Type	Execution Event Type	Notes
beforeLoad	<p><i>type</i> : the read operation type</p> <ul style="list-style-type: none"> <li>■ create</li> <li>■ edit</li> <li>■ view</li> <li>■ copy</li> <li>■ print</li> <li>■ email</li> <li>■ quickview</li> </ul> <p><i>form</i> : an <code>nlobjForm</code> object representing the current form</p> <p><i>request</i> : an <code>nlobjRequest</code> object representing the GET request (Only available for browser requests.)</p>	<p>Event occurs whenever a read operation on a record occurs. These operations include navigating to a record in the UI, reading a record in web services, attaching a child custom record to its parent, detaching a child custom record from its parent, or calling <code>napiLoadRecord</code>. The function cannot be used to source standard records. Use the <code>pagelInit</code> client script for this purpose. See <a href="#">Client Event Types</a>.</p> <p>The user-defined function is executed prior to returning the record or page. The function is passed either the <i>type</i>, <i>form</i>, or <i>request</i> arguments by the system.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p><b>Note:</b> beforeLoad user events cannot be triggered when you load/access an online form.</p> </div>
beforeSubmit	<p><i>type</i> : the write operation type</p> <ul style="list-style-type: none"> <li>■ create</li> <li>■ edit</li> <li>■ delete</li> <li>■ xedit - (see <a href="#">Inline Editing and SuiteScript</a>)</li> <li>■ approve - (only available for certain record types)</li> </ul>	<p>Events on a beforeSubmit operation occur prior to any write operation on the record. Changes to the current record at this stage will be persisted during the write operation.</p> <p>The beforeSubmit operation is useful for validating the submitted record, performing any restriction and permission checks, and performing any last-minute changes to the current record.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>■ The <i>approve</i>, <i>cancel</i>, and <i>reject</i> argument types are only available for record types such as sales</li> </ul>

Operation Type	Execution Event Type	Notes
	<ul style="list-style-type: none"> <li>■ reject - (only available for certain record types)</li> <li>■ cancel - (only available for certain record types)</li> <li>■ pack- (only available for certain record types, for example Item Fulfillment records)</li> <li>■ ship - (only available for certain record types, for example Item Fulfillment records)</li> <li>■ markcomplete (specify this type for a beforeSubmit script to execute when users click the Mark Complete link on call and task records)</li> <li>■ reassigned (specify this type for a beforeSubmit script to execute when users click the Grab link on case records)</li> <li>■ editforecast (specify this type for a beforeSubmit script to execute when users update opportunity and estimate records using the Forecast Editor)</li> </ul>	<p>orders, expense reports, timebills, purchase orders, and return authorizations.</p> <ul style="list-style-type: none"> <li>■ Only beforeLoad and afterSubmit user event scripts will execute on the Message record type when a message is created by an inbound email case capture. Scripts set to execute on a beforeSubmit event will not execute.</li> <li>■ User Event Scripts cannot override custom field permissions. For instance, if a user's role permissions and a custom field's permissions differ, beforeSubmit cannot update the custom field, even if the script is set to execute as Administrator.</li> <li>■ Attaching a child custom record to its parent or detaching a child custom record from its parent will trigger an edit event.</li> </ul> <p><b>Best practices:</b> To set a field on a record or make ANY changes to a record that is being submitted, do so on a <b>beforeSubmit</b> operation, NOT an <b>afterSubmit</b> operation. If you set a field on an <b>afterSubmit</b>, you will be duplicating a record whose data has already been committed to the database.</p> <div style="border: 1px solid #f0e68c; padding: 10px; margin-top: 10px;"> <p> <b>Important:</b> Do not attempt to load or submit the current record with nlapiLoadRecord or nlapiSubmitRecord on a beforeSubmit event. Doing so will result in the loss of data.</p> </div>
afterSubmit	<p><b>type :</b> the write operation type</p> <ul style="list-style-type: none"> <li>■ create</li> <li>■ edit</li> <li>■ delete</li> <li>■ xedit - (see <a href="#">Inline Editing and SuiteScript</a>)</li> <li>■ approve - (only available for certain record types)</li> <li>■ cancel - (only available for certain record types)</li> <li>■ reject - (only available for certain record types)</li> <li>■ pack - (only available for certain record types, for example Item Fulfillment records)</li> </ul>	<p>Events on an <b>afterSubmit</b> operation occur immediately following any write operation on a record.</p> <div style="background-color: #e0f2ff; border: 1px solid #80bfff; padding: 10px; margin-top: 10px;"> <p> <b>Note:</b> Asynchronous <b>afterSubmit</b> user events are only supported during webstore checkout.</p> </div> <p>The <b>afterSubmit</b> operation is useful for performing any actions that need to occur following a write operation on a record. Examples of these actions include email notification, browser redirect, creation of dependent records, and synchronization with an external system.</p> <p>Attaching a child custom record to its parent or detaching a child custom record from its parent will trigger an edit event.</p>

Operation Type	Execution Event Type	Notes
	<ul style="list-style-type: none"> <li>■ ship - (only available for certain record types, for example Item Fulfillment records)</li> <li>■ dropship - (for purchase orders with items specified as "drop ship")</li> <li>■ specialorder - (for purchase orders with items specified as "special order")</li> <li>■ orderitems - (for purchase orders with items specified as "order item")</li> <li>■ paybills - (use this type to trigger afterSubmit user events for Vendor Payments from the Pay Bill page. Note that no sublist line item information will be available. Users must do a lookup/search to access line item values.)</li> </ul>	<p><b>Note:</b> The <i>approve</i>, <i>cancel</i>, and <i>reject</i> argument types are only available for record types such as sales orders, expense reports, timebills, purchase orders, and return authorizations.</p> <p><b>Best Practices:</b> Users should be doing post-processing of the current record on an afterSubmit. Use Case:</p> <ol style="list-style-type: none"> <li>1. Load the record you want to make changes to by calling the nlapiLoadRecord API. Do NOT load the record object by using nlapiGetRecord, as this API returns the record in READ ONLY mode; therefore, changes made to the record cannot be accepted and an error is thrown.</li> <li>2. After using nlapiLoadRecord to load the record, make the changes to the record, and submit the record.</li> </ol> <p>For example, if you have a sales order that you want tied to an estimate, load the sales order record, update it with any information, and submit it again.</p>

## How Many User Events Can I Have on One Record?

There is no limit to the number of user event scripts you can execute on a particular record type. For example, you could have 10 beforeLoad, 9 beforeSubmit, and 15 afterSubmit executing functions on a Customer record. However, assigning this many executable functions to one record type is **highly discouraged**, as this could negatively affect user experience with that record type. In other words, if you have 10 beforeLoad scripts that must complete their execution before a record loads into the browser for the user, this may significantly increase the time it takes for the record to load. As a consequence, the user's experience working with the record will be negatively affected.

Developers who include scripts in their bundles should also be aware of the number of user events scripts that might already be deployed to records types in the target account. For example, if 8 beforeSubmit user event scripts are deployed to the Sales Order record in the target account, and your bundle includes another 7 beforeSubmit user event scripts on the Sales Order record type, this is 15 beforeSubmit scripts running every time a user clicks Save on the record. Although all of the scripts will run, the time it takes for the record to actually save may be significantly increased, again, negatively affecting user experience with the record.

If you must run multiple user event scripts on one record type, and are experiencing very slow execution times, use the Script Performance Monitor SuiteApp to troubleshoot the issue. The Script Performance Monitor keeps 30 days worth of performance data for each scriptable record type. You can view the overall execution time of each record instance, as well as the execution time of

each individual script. See the help topic [Application Performance Management \(APM\)](#) for additional information.

## Running a User Event Script in NetSuite

To run a user event script in NetSuite, you must:

1. Create a JavaScript file for your user event script.
2. Load the file into NetSuite.
3. Create a Script record.
4. Define all runtime options on the Script Deployment page.

If you are new to SuiteScript and need information on each of these steps, see [Running Scripts in NetSuite Overview](#).

## User Event Script Samples

The following samples are provided in this section:

- Generating a Record Log
- Creating Follow-up Phone Call Records for New Customers
- Enhancing NetSuite Forms with User Event Scripts

## Generating a Record Log

This user event script creates an execution log entry containing the type, record type, and internalId of the current record.

```
function beforeLoad(type,form)
{
    var newId = nlapiGetRecordId();
    var newType = nlapiGetRecordType();
    nlapiLogExecution('DEBUG','<Before Load Script> type:'+type+, RecordType: '+newType+, Id:' +newId);
}
function beforeSubmit(type)
{
    var newId = nlapiGetRecordId();
    var newType = nlapiGetRecordType();
    nlapiLogExecution('DEBUG','<Before Submit Script> type:'+type+, RecordType: '+newType+, Id: '+newId);
}
function afterSubmit(type)
{
    var newId = nlapiGetRecordId();
    var newType = nlapiGetRecordType();
    nlapiLogExecution('DEBUG','<After Submit Script> type:'+type+, RecordType: '+newType+, Id:
```



```
'+newId);
}
```

## Creating Follow-up Phone Call Records for New Customers

A CRM use case that could be addressed with user event scripts is creating a follow-up phone call record for every newly created customer record. The solution is to deploy a script on the customer record's *afterSubmit* event that will create the phone call record.

In the above use case, *afterSubmit* is a better event to handle the logic than *beforeSubmit*. In the *beforeSubmit* event, the customer data has not yet been committed to the database. Hence, putting the phone call logic in the *afterSubmit* event guarantees there will not be an orphan phone call record.



**Note:** During design time, developers should carefully consider in which event to implement their server logic.

```
function followUpCall_CustomerAfterSubmit(type)
{
    //Only execute the logic if a new customer is created
    if (type == 'create')
    {
        //Obtain a handle to the newly created customer record
        var custRec = nlapiGetNewRecord();

        if (custRec.getFieldValue('salesrep') != null )
        {
            //Create a new blank instance of a Phone Call
            var call = nlapiCreateRecord("phonecall");

            //Setting the title field on the Phone Call record
            call.setFieldValue('title', 'Make follow-up call to new customer');

            //Setting the assigned field to the sales rep of the new customer
            call.setFieldValue('assigned', custRec.getFieldValue('salesrep'));

            //Use the library function to obtain a date object that represents tomorrow
            var today = new Date();
            var tomorrow = nlapiAddDays(today, 1);
            call.setFieldValue('startdate', nlapiDateToString(tomorrow));

            //Setting the phone field to the phone of the new customer
            call.setFieldValue('phone', custRec.getFieldValue('phone'));

            try
            {
                //Committing the phone call record to the database
                var callId = nlapiSubmitRecord(call, true);
                nlapiLogExecution('DEBUG', 'call record created successfully', 'ID = ' + callId);
            }
            catch (e)
```

```

    {
      nlapiLogExecution('ERROR', e.getCode(), e.getDetails());
    }
  }
}
}

```



**Note:** APIs such as `nlapiSubmitRecord` that access the database should be wrapped in try-catch blocks.

The phone call use case may be further enhanced by redirecting the user to the Phone Call page after it has been created. This redirect is accomplished by putting in redirect logic after the phone call record is submitted.

```

try
{
  //committing the phone call record to the database
  var callId = nlapiSubmitRecord(call, true);
  nlapiLogExecution('DEBUG', 'call record created successfully', 'ID = ' + callId);

  //Redirect the user to the newly created phone call
  nlapiSetRedirectURL('RECORD', 'phonecall', callId, false, null);
}
catch (e)
{
  nlapiLogExecution('ERROR', e.getCode(), e.getDetails());
}

```

User event scripts are not only triggered by user actions carried out through the browser, they are also triggered by other means as well (for example, CSV or Web Services).

Examples:

- Using CSV to import records triggers *before submit* and *after submit* events
- Using the SuiteTalk “GET” operation to retrieve an existing record would trigger its *before load* event. Sometimes these events invoke scripts not designed to be executed in that manner and create undesirable results. To prevent a script from getting executed by the wrong execution context, use the `nlobjContext` object as a filter.

For example, to ensure a *before load* user event script is executed only when a record is created using the browser interface, the script must check both the type argument and the execution context as (as shown below):

```

function myBeforeLoadUE(type)
{
  //obtain the context object
  var context = nlapiGetContext();
  if (type == 'create' && context.getExecutionContext == 'userinterface')
  {...}
}

```

Note that the API `nlapiGetContext()` is not exclusive to user event scripts. It can also be used in [Client Scripts](#) and [Suitelets](#).



**Note:** The nlobjContext object provides metadata of the script's context. Use this information to help implement fine-grained control logic in SuiteScript.

## Enhancing NetSuite Forms with User Event Scripts

Another common use of user event scripts is to dynamically customize or enhance entry forms and transaction forms. This approach gives NetSuite forms the ability to customize themselves in runtime – something that cannot be done with pre-configured, roles-based forms.

In NetSuite, entry forms and transaction forms are customized by administrators. The placement of UI elements (fields, tabs, sublists) on a form can be arranged, or be made inline or hidden depending on the business needs of the end users. Multiple forms can be created for a record type and assigned to specific roles. Typically this kind of customization is done during design time. Custom forms are confined to specific roles and do not allow for a lot of runtime customization. A user event script on a record's **beforeLoad** event can provide flexibility to runtime customization.



**Note:** For more specific information about NetSuite entry forms and transaction forms, see the help topic [Custom Forms](#) in the NetSuite Help Center.

The key to using user event scripts to customize a form during runtime is a second argument named **form** in the **beforeLoad** event. This optional argument is the reference to the entry/transaction form. Developers can use this to dynamically change existing UI elements, or add new ones. The UI elements are added using the [UI Objects API](#).

A use case for this scripting capability could be the following:

To improve month-end sales, a company introduces an end-of-month promotion that is only active for the last five days of the month. All sales order forms must have a custom field called “Eligible EOM Promotion” on the last five days of the month.

The following is a sample user event script that is meant to be deployed on the **beforeLoad** event of the sales order record.

```
*****
 *This function is a module to implement at end of
 * month(last 5 days of month) promotion for sales
 * orders. It is meant to be deployed on the before
 * load event of the sales order record.
 */
function customizeUI_SalesOrderBeforeLoad(type, form)
{
    var currentContext = nlapiGetContext();

    //Execute the logic only when creating a sales order with the browser UI
    if (type == 'create' && currentContext.getExecutionContext() == 'userinterface')
    {
        var fieldId = 'custpage_eom_promotion';
        var fieldLabel = 'Eligible EOM promotion';
        var today = new Date();
        var month = today.getMonth();
        var date = today.getDate();
        nlapiLogExecution('DEBUG', 'month date', month + '' + date);

        //February
    }
}
```

```

if (month==1)
{
    if (date==24 | date==25 | date==26 | date==27 | date==28 | date==29)
        form.addField(fieldId, 'checkbox', fieldLabel);
}
//31-day months
elseif (month==0 | month==2 | month ==4 | month==6 | month==7 | month==9 | month==11)
{
    if ( date==27 | date==28 | date==29 | date==30 | date==31)
        form.addField(fieldId, 'checkbox', fieldLabel);
}
//30-day months
else
{
    if ( date==26 | date==27 | date==28 | date==29 | date==30)
        form.addField(fieldId, 'checkbox', fieldLabel);
}
}
}
}

```

When the script is deployed, all sales order forms will display the **Eligible EOM Promotion** check box during the last five days of the month.

The screenshot shows a Sales Order form with the following details:

- Primary Information:**
  - WOLFE ELECTRONICS SALES ORDER \*
  - MEMO
  - ORDER # SORD100440
  - CUSTOMER \* <Type then tab>
  - DATE \* 12/29/2014
  - STATUS \* Pending Approval
  - PO #
- Summary:**

Summary	
TOTAL (ALT. SALES)	0.00
SUBTOTAL	0.00
DISCOUNT ITEM	0.00
GST/HST	0.00
PST	0.00
SHIPPING COST	
HANDLING COST	
GIFT CERTIFICATE	0.00
<b>TOTAL</b>	<b>0.00</b>
- Sales Information:**
  - OPPORTUNITY
  - EXCLUDE COMMISSIONS
- Classification:**
  - DEPARTMENT
  - CLASS
  - SALES REP PHONE
  - LOCATION East Coast
  - ELIGIBLE EOM PROMOTION

Note that since these UI elements are created dynamically, they are superficial and do not have supporting back end data models. There is a disconnect between the UI and back end data, hence the script-created fields' values **will not be saved**.

UI elements (such as the Eligible EOM promotion field) created with user event scripts and [SuiteScript Objects](#) are scriptable by client script APIs. A remedy to the disconnect problem is linking the script-created field to a real field (with back end data support) via a client script. The value of the real field, which might be made hidden or inline on the form definition, is driven by the value entered in the script-created field. The real fields are populated and the data is saved.

# Suitelets

The following topics are covered in this section. If you are not familiar with Suitelets, these topics should be read in order.

- What Are Suitelets?
- Suitelet Script Execution
- Building Custom Workflows with Suitelets
- Building Suitelets with UI Objects
- Backend Suitelets
- Reserved Parameter Names in Suitelet URLs
- SuiteScript and Externally Available Suitelets
- Running a Suitelet in NetSuite
- Suitelets Samples

## What Are Suitelets?

Suitelets are extensions of the SuiteScript API that give developers the ability to build custom NetSuite pages and backend logic. Suitelets are server-side scripts that operate in a request-response model. They are invoked by HTTP GET or POST requests to system generated URLs.

**i Note:** Suitelets are not intended to work inside web stores. Use online forms to embed forms inside a web store.

Shown below are screenshots of a Suitelet with a few fields. The Suitelet is invoked by making a GET request from the browser. Notice that this Suitelet is built with SuiteScript [UI Objects](#), which encapsulate scriptable interface components that have a NetSuite look-and-feel.

After a Suitelet has been deployed, developers can create NetSuite tasklinks to these scripts, which can then be used to customize existing NetSuite centers.

The screenshot shows a Suitelet interface titled "Suitelet - GET call". At the top, there is a navigation bar with links: Activities, Payments, Transactions, Lists, Reports, Customization, Documents, Setup (selected), Training, and Support. Below the title, there is a "Submit" button. The interface contains three input fields: a "TEXT FIELD" containing the text "The quick brown dog jumped over the lazy fox.", a "SELECT FIELD" dropdown containing the value "Abe Simpson", and an "INTEGER FIELD" containing the value "4,000".

When the Submit button is clicked, the same Suitelet is invoked again with a HTTP POST event. The values entered in the previous screen are displayed in inline (read-only) mode.

The screenshot shows a Suitelet interface titled "Suitelet - POST call". At the top, there is a navigation bar with links: Activities, Payments, Transactions, Lists, Reports, Customization, Documents, Setup, Training, and Support. Below the title, there is a message "TEXT FIELD VALUE ENTERED: The quick brown dog jumped over the lazy fox.". To the right, there is a message "SELECT FIELD VALUE ENTERED: Abe Simpson". Below these messages, there is another message "INTEGER FIELD VALUE ENTERED: 4,000".

Below is the source code for this Suitelet. It is executed on the server, which generates HTML and sends it to the browser.

```

function gettingStartedSuitelet(request, response)
{
    if (request.getMethod() == 'GET' )
    {
        //Create the form and add fields to it
        var form = nlapiCreateForm("Suitelet - GET call");
        form.addField('custpage_field1', 'text', 'Text Field' ).setDefaultValue('This is a text file');
        form.addField('custpage_field2', 'integer', 'Integer Field' ).setDefaultValue(10);
        form.addField('custpage_field3', 'select', 'Select Field', 'customer' );

        form.addSubmitButton('Submit' );

        response.writePage(form);
    }
    //POST call
    else
    {
        var form = nlapiCreateForm("Suitelet - POST call");

        //create the fields on the form and populate them with values from the previous screen
        var resultField1 = form.addField('custpage_res1', 'text', 'Text Field value entered: ' );
        resultField1.setDefaultValue(request.getParameter('custpage_field1' ));
        resultField1.setDisplayType('inline' );

        var resultField2 = form.addField('custpage_res2', 'integer', 'Integer Field value entered: ' );
        resultField2.setDefaultValue(request.getParameter('custpage_field2' ));
        resultField2.setDisplayType('inline' );

        var resultField3 = form.addField('custpage_res3', 'select', 'Select Field value entered: ', 'customer' );
        resultField3.setDefaultValue(request.getParameter('custpage_field3' ));
        resultField3.setDisplayType('inline' );

        response.writePage(form);
    }
}

```

The entry point of the function has two mandatory arguments: `request` and `response`. These arguments are instances of `nlobjRequest` and `nlobjResponse`, respectively.

Typically, invoking a Suitelet via a browser would make a HTTP GET call. The type of HTTP call is determined by the `nlobjRequest.getMethod()` API. The code creates an `nlobjForm` object and populates it with SuiteScript `UI Objects`. The populated form is sent to the `response` object via the `nlobjResponse.writePage(pageobject)` API.

When the user clicks the Submit button, an HTTP POST call is made. The code's `else` block obtains the values entered in the first page from the `request` object and populates them into another `nlobjForm` object and sends it to `response.writePage(pageobject)`.



**Note:** The JavaScript alert function is not supported in server-side scripts, such as Suitelets.

```
function alert_test ()  
{  
    alert('Hello World');  
}
```

## Suitelet Script Execution

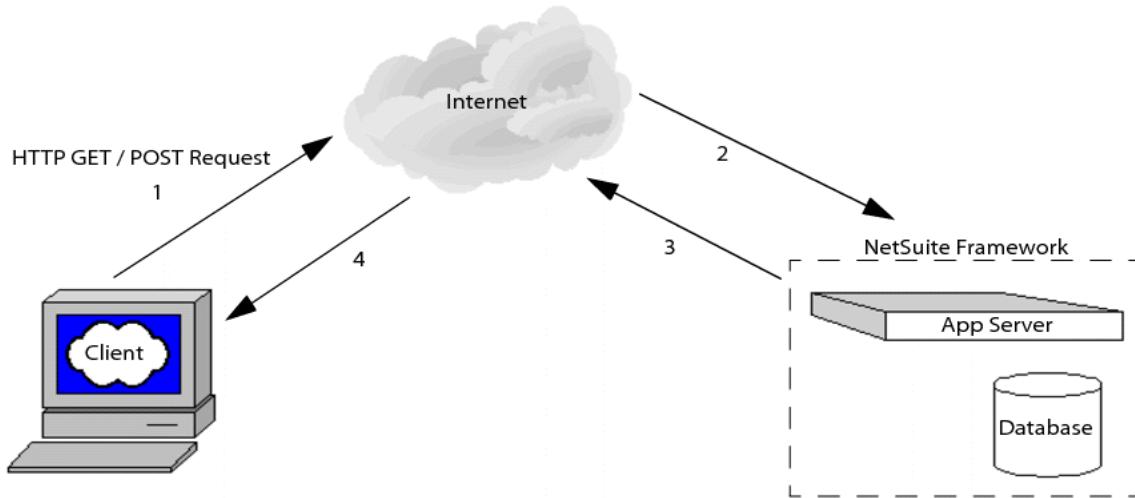
The following steps and diagram provide an overview of the Suitelet execution process:

1. Client initiates an HTTP GET or POST request (typically from a browser) for a system-generated URL. A web request object ([nlobjRequest](#)) contains the data from the client's request.
2. The user's script is invoked, which gives the user access to the entire Server SuiteScript API as well as a web request and web response object.
3. NetSuite processes the user's script and returns a web response object ([nlobjResponse](#)) to the client. The response can be in following forms:
  - Free-form text
  - HTML
  - RSS
  - XML
  - A browser redirect to another page on the Internet



**Important:** You can only redirect to external URLs from Suitelets that are accessed externally (in other words, the Suitelet has been designated as "Available Without Login" and is accessed from its external URL).

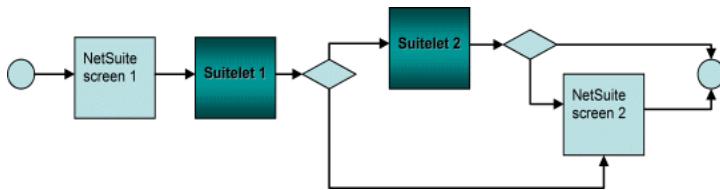
- A browser redirect to an internal NetSuite page. The NetSuite page can be either a standard page or custom page that has been dynamically generated using [UI Objects](#).
4. The data renders in the user's browser.



# Building Custom Workflows with Suitelets

With the user event scripts, Suitelets, and UI Objects, SuiteScript developers can create custom workflows by chaining together standard and/or custom NetSuite pages. These workflows may be complemented by custom backend logic.

The following diagram shows how a SuiteScript developer can potentially create a workflow that starts off with either a standard or custom NetSuite record, then redirects to a Suitelet, then redirects to either another Suitelet or standard/custom NetSuite record, all depending on the logic of the developer's application.



## Building Suitelets with UI Objects

When building Suitelets, developers can use SuiteScript [UI Objects](#) to create custom pages that look like NetSuite pages. SuiteScript UI objects encapsulate the elements for building NetSuite-looking portlets, forms, fields, sublists, tabs, lists, and columns.

When developing a Suitelet with UI objects, you can also add custom fields with inline HTML.

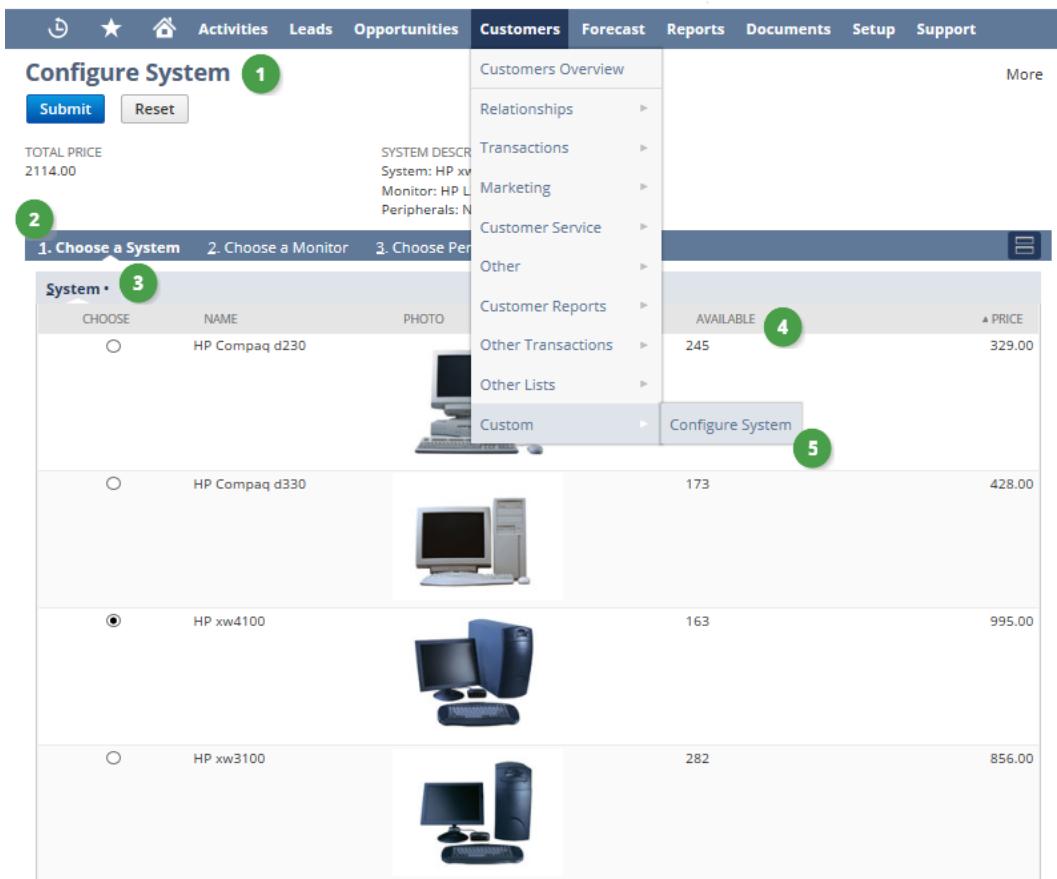


**Important:** When adding UI elements to an [existing NetSuite page](#), you must prefix the object name with `custpage`. This minimizes the occurrence of field/object name conflicts. For example, when adding a custom tab to an entry form, the name should follow a convention similar to `custpage customtab` or `custpage mytab`.

The figure below shows a custom interface that has been built with the following SuiteScript UI objects:

- `nlobjForm`
- `nlobjTab`
- `nlobjSubList`
- `nlobjField`

Note that a custom menu link was created to access the Suitelet. In this figure, the Configure System Suitelet can be accessed by going to Customers > Custom > Configure System. For information on creating menu links for Suitelets, see [Running a Suitelet in NetSuite](#).



Item	Description
1	SuiteScript UI Object: FORM
2	SuiteScript UI Object: TAB
3	SuiteScript UI Object: SUBLIST
4	SuiteScript UI Object: FIELD
5	Custom Menu Link

## Backend Suitelets

Suitelets give developers the ability to build custom NetSuite pages. However, developers can create Suitelets that do not generate any UI elements. These kinds of Suitelets are referred to as backend Suitelets. Their sole purpose is to execute backend logic, which can then be parsed by other parts of a custom application.

Just like a Suitelet that builds NetSuite pages, a backend Suitelet is invoked by making HTTP GET or POST calls to a NetSuite-generated Suitelet URL.

The following are good uses of backend Suitelets:

- Providing a service for backend logic to other SuiteScripts, or to other external hosts outside of NetSuite

- Offloading server logic from client scripts to a backend Suitelet shipped without source code to protect sensitive intellectual property



**Important:** RESTlets can provide an alternative to backend Suitelets. For general information about this type of script, see [RESTlets](#). For a comparison, see [RESTlets Compared to Suitelets](#).

A use case of a backend Suitelet is a service that provides customer information based on a phone number. The following is the code for a Suitelet that returns customer entity IDs (for records with matching phone numbers) separated by the | character.

```
*****
*This function searches for customer records* that match a supplied parameter custparam_phone
*and return the results in a string separated* by the | character.
*/function lookupPhoneBackendSuitelet(request, response)
{
  if (request.getMethod() == 'GET' )
  {
    //null check on the required parameter if (request.getParameter('custparam_phone') != null )
    {
      //Setting up the filters and columns var filters = newArray ();
      var columns = newArray ();

      //Use the supplied custparam_phone value as filter
      filters[0] = new nlobjSearchFilter('phone', null, 'is', request.getParameter('custparam_p
hone'));
      columns[0] = new nlobjSearchColumn('entityid', null, null );

      //Search for customer records that match the filters var results = nlapiSearchRecord('custome
r', null, filters, columns);

      if (results != null )
      {
        var resultString = " ";
        //Loop through the results for (var i = 0; i < results.length ; i++)
        {
          //constructing the result string var result = results[i];
          resultString = resultString + result.getValue('entityid');

          //adding the| separator if (i != parseInt (results.length - 1))
          {
            resultString = resultString + '|';
          }
          nlapiLogExecution('DEBUG', 'resultString', resultString);
        }
        response.write(resultString);
      }
      else
      {
        response.write('none found');
      }
    }
  }
}
```

```
}
```

Notice that this Suitelet does not use any UI Object APIs. Communication with the Suitelet is done strictly with the request and response objects. NetSuite generates a URL to invoke this Suitelet. To correctly invoke it, the `custparam_phone` value (bold) needs to be appended at the end of the invoking URL:

```
https://system.netsuite.com/app/site/hosting/scriptlet.nl?script=6&deploy=1&custparam_phone=(12  
3)-456-7890
```

The code that calls this backend Suitelet needs to do the following:

1. Use `nlapiResolveURL` to dynamically obtain the invoking URL
2. Supply required parameters
3. Process the returned results



**Note:** Backend Suitelets should not be used to get around SuiteScript usage governance. Suitelets designed with this intention are considered abusive by NetSuite.

## Reserved Parameter Names in Suitelet URLs

Certain names are reserved and should not be referenced when naming custom parameters for Suitelet URLs.

The following table contains a list of reserved parameter names:

Reserved Suitelet URL Parameter Names

e	print
id	email
cp	q
I	si
popup	st
s	r
d	displayonly
_nodrop	nodisplay
sc	deploy
sticky	script

If any of your parameters are named after any of the reserved parameter names, your Suitelet may throw an error saying, "There are no records of this type." To avoid naming conflicts, NetSuite recommends that all custom URL parameters are prefixed with `custom`. For example, use `custom_id` instead of `id`.

## SuiteScript and Externally Available Suitelets

Only a subset of the SuiteScript API is supported in **externally available** Suitelets (Suitelets set to Available Without Login on the Script Deployment page). For a list of these APIs, in the NetSuite Help Center see these topics related to online forms.

- Working with Online Forms
- Why are only certain APIs supported on online forms?



**Note:** The same concepts that apply to online forms also apply to externally available Suitelets.

Note that if you want to use all available SuiteScript APIs in a Suitelet, your Suitelet will require a valid NetSuite session. (A valid session means that users have authenticated to NetSuite by providing their email address and password.)

On the Script Deployment page, leave the Available Without Login check box unselected if you want to deploy a Suitelet that requires a valid session. (See also [Setting Available Without Login](#) for more information on this runtime option.)

Script Deployment	
<b>SCRIPT</b>	Simple List Suitelet
<b>TITLE</b>	Simple List Suitelet
<b>ID</b>	customdeploy1
<input checked="" type="checkbox"/> <b>DEPLOYED</b>	
<b>STATUS</b>	Testing
<b>EVENT TYPE</b>	
<b>LOG LEVEL</b>	Debug
<b>EXECUTE AS ROLE</b>	Administrator
<input type="checkbox"/> <b>AVAILABLE WITHOUT LOGIN</b>	
<b>URL</b>	
/app/site/hosting/scriptlet.nl?script=130&deploy=1	



**Important:** [UI Objects](#) can be used without a valid session. Therefore, they are supported in externally available Suitelets.

## Running a Suitelet in NetSuite

To run a Suitelet in NetSuite, you must:

1. Create a JavaScript file for your Suitelet.
2. Load the file into NetSuite.
3. Create a Script record.
4. Define all runtime options on the Script Deployment page.

If you are new to SuiteScript and need information on each of these steps, see [Running Scripts in NetSuite Overview](#).

Note that if you want users to be able to access/launch a Suitelet from the UI, you can create a menu item for the Suitelet.

The following figure shows the Links tab on the Script Deployment page for a Suitelet. Select the Center where the link to the Suitelet will be accessible (for example, Customer Center, Vendor Center, etc). Next, set the Section (top-level menu tab) in the Center, then choose a category under the section. Finally, create a UI label for the link. Be sure to click Add when finished.



**Note:** The Classic Center is a default center. It is not specific to customers, partners, or vendors.

CENTER*	SECTION*	CATEGORY	LABEL	INSERT BEFORE
Classic Center	Support	Customer Service	Contact Support Rep	

When the Script Deployment page is saved, a link to the Suitelet appears (see figure).

SCRIPT	EVENT TYPE	LOG LEVEL	EXECUTE AS ROLE	AVAILABLE WITHOUT LOGIN	URL
Folder example		Debug	Administrator	<input type="checkbox"/>	/app/site/hosting/scriptlet.nl?script=69&deploy=1
TITLE					
Folder example					
ID					
customdeploy1					
<input checked="" type="checkbox"/> DEPLOYED					
STATUS					
Testing					

## Suitelets Samples

The following sample Suitelets show how to return HTML and XML documents, as well as how to create forms and lists.

- Writing Your First Suitelet
- Return a Simple XML Document
- Create a Simple Form
- Create a Simple List
- Add a Suitelet to a Tab
- Create a Suitelet Email Form
- Create a Form with a URL Field
- Using Embedded Inline HTML in a Form

## Writing Your First Suitelet

This basic Hello World! sample shows how to return an HTML document in a Suitelet.

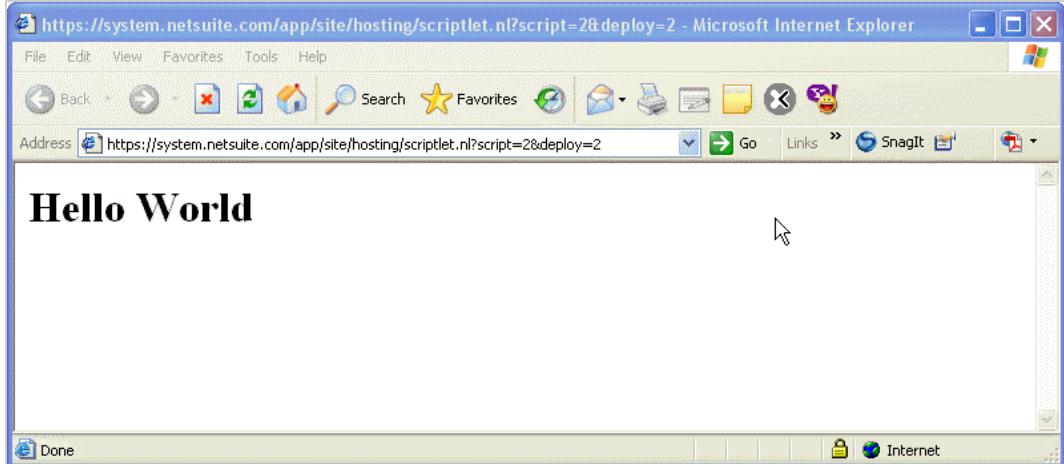
### Script:

```
function demoHTML(request, response)
{
```

```

var html = '<html><body><h1>Hello World</h1></body></html>';
response.write( html );
//prefix header with Custom-Header. See nlobjResponse.
setHeader(name, value)
  response.setHeader('Custom-Header-Demo', 'Demo');
}

```



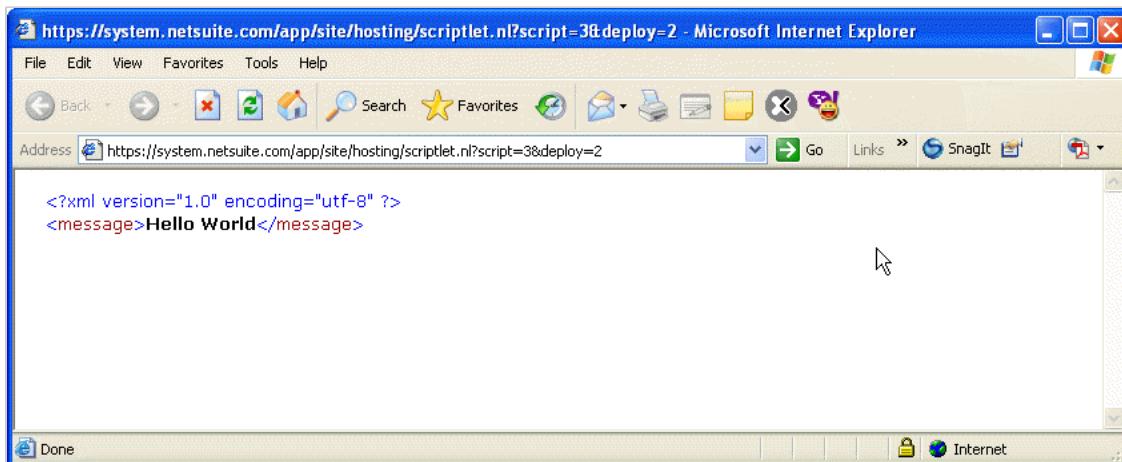
## Return a Simple XML Document

### Script:

```

function demoXML(request, response)
{
  var xml = '<?xml version="1.0" encoding="utf-8" ?>' +
    '<message>'+'Hello World'+ '</message>';
  response.write( xml );
  response.setHeader('Custom-Header-Demo', 'Demo');
}

```



## Create a Simple Form

This form is generated using the [nlobjForm](#) UI object. Note that Suitelets built with the SuiteScript [UI Objects](#) API can be accessed without a valid session. In other words, Suitelets using UI objects can be set to **Available Without Login** on the Script Deployment page. (For information on deploying scripts, see [Step 5: Define Script Deployment](#).)

### Script:

```
function demoSimpleForm(request, response)
{
    if ( request.getMethod() == 'GET' )
    {
        var form = nlapiCreateForm('Simple Form');
        var field = form.addField('textfield','text', 'Text');
        field.setLayoutType('normal', 'startcol')
        form.addField('datefield','date', 'Date');
        form.addField('currencyfield','currency', 'Currency');
        form.addField('textareafield','textarea', 'Textarea');

        var select = form.addField('selectfield', 'select', 'Select');
        select.addSelectOption("", "");
        select.addSelectOption('a', 'Albert');
        select.addSelectOption('b', 'Baron');
        select.addSelectOption('c', 'Chris');
        select.addSelectOption('d', 'Drake');
        select.addSelectOption('e', 'Edgar');

        var sublist = form.addSubList('sublist','inlineeditor', 'Inline Editor Sublist');
        sublist.addField('sublist1', 'date', 'Date');
        sublist.addField('sublist2', 'text', 'Text');
        sublist.addField('sublist3', 'currency', 'Currency');
        sublist.addField('sublist4', 'textarea', 'Large Text');
        sublist.addField('sublist5', 'float', 'Float');

        form.addSubmitButton('Submit');

        response.writePage( form );
    }
    else
        dumpResponse(request,response);
}
```

## Create a Simple List

This list is generated using the `nlobjList` UI object. Note that Suitelets built with the [UI Objects API](#) can be accessed without a valid session. In other words, Suitelets using UI objects can be set to **Available Without Login** on the Script Deployment page. (For information on deploying scripts, see [Step 5: Define Script Deployment](#).)



**Important:** If your browser is inserting scroll bars in this code sample, maximize your browser window, or expand the main frame that this sample appears in.

### Script:

```
function demoList(request, response)
{
    var list = nlapiCreateList('Simple List');

    // You can set the style of the list to grid, report, plain, or normal, or you can get the
    // default list style that users currently have specified in their accounts.
    list.setStyle(request.getParameter('style'));

    var column = list.addColumn('number', 'text', 'Number', 'left');
    column.setURL(nlapiResolveURL('RECORD','salesorder'));
    column.addParamToURL('id','id', true);

    list.addColumn('trandate', 'date', 'Date', 'left');
    list.addColumn('name_display', 'text', 'Customer', 'left');
    list.addColumn('salesrep_display', 'text', 'Sales Rep', 'left');
    list.addColumn('amount', 'currency', 'Amount', 'right');
```

```

var returncols = new Array();
returncols[0] = new nlobjSearchColumn('trandate');
returncols[1] = new nlobjSearchColumn('number');
returncols[2] = new nlobjSearchColumn('name');
returncols[3] = new nlobjSearchColumn('salesrep');
returncols[4] = new nlobjSearchColumn('amount');

var results = nlapiSearchRecord('estimate', null, new nlobjSearchFilter('mainline',null,'i
s','T'), returncols);
list.addRows( results );

list.addPageLink('crosslink', 'Create Phone Call', nlapiResolveURL('TASKLINK','EDIT_CALL')
);
list.addPageLink('crosslink', 'Create Sales Order',
nlapiResolveURL('TASKLINK','EDIT_TRAN_SALESORD'));

list.addButton('custombutton', 'Simple Button', "alert('Hello World')");
response.writePage( list );
}

```

The screenshot shows a Suitelet interface with a top navigation bar containing icons for Home, Activities, Payments, Transactions, Lists, Reports, Customization, Documents, Setup, Training, and Support. Below the navigation bar, the title 'Simple List' is displayed. To the right of the title are two links: 'Create Phone Call' and 'Create Sales Order'. A button labeled 'Simple Button' is visible. The main content area contains a table with the following data:

NUMBER	DATE	CUSTOMER	SALES REP	AMOUNT
EST10001	2/8/2003	Abe Simpson	Jon Baker	14,017.00
EST10002	4/1/2003	Elijah Tuck	Poncho Ponchereli	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 Ponchereli	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

## Add a Suitelet to a Tab

The following user event script shows how to add a tab to a record, and then on the tab, add a Suitelet. In this example a tab called *Sample Tab* is added to a Case record. A link is added to the Sample Tab that, when clicked, opens a Suitelet.



**Important:** If your browser is inserting scroll bars in this code sample, maximize your browser window, or expand the main frame that this sample appears in.

### Script:

```

/*Create a user event beforeLoad function that takes type and form as parameters.
 *Later you will define the script execution context by providing a value for type.
 *The form argument instantiates a SuiteScript nlobjForm object, which allows you
 *to add fields and sublists later on in the script.
 */
function beforeLoadTab(type, form)
{
var currentContext = nlapiGetContext();
var currentUserID = currentContext.getUser();

/*Define the value of the type argument. If the Case record is edited or viewed,
 *a tab called Sample Tab is added. Note that the script execution context is set to
 *userinterface. This ensures that this script is ONLY invoked from a user event
 *occurring through the UI.
*/
if( (currentContext.getExecutionContext() == 'userinterface') && (type == 'edit' | type == 'view'))
{
    var SampleTab = form.addTab('custpage_sample_tab', 'SampleTab');

//On Sample Tab, create a field of type inlinehtml.
var createNewReqLink = form.addField('custpage_new_req_link','inlinehtml', null, null,
'custpage_sample_tab');

//Define the parameters of the Suitelet that will be executed.
var linkURL = nlapiResolveURL('SUITELET', 'customscript12','customdeploy1', null)
+ '&customerid=' + nlapiGetRecordId();

//Create a link to launch the Suitelet.
createNewReqLink.setDefaultValue('<B>Click <A HREF=' + linkURL +'>here</A> to
create a new document signature request record.</B>');

//Add a sublist to Sample Tab.
var signatureRequestSublist = form.addSubList('custpage_sig_req_sublist',
'list', 'Document Signature Requests','custpage_sample_tab');

signatureRequestSublist.addField('custpage_req_name', 'text', 'Name');
signatureRequestSublist.addField('custpage_req_status', 'text', 'Status');
signatureRequestSublist.addField('custpage_req_created', 'date', 'Date
Created');
}
}

```

The screenshot shows the NetSuite Case form interface. At the top, there are buttons for Save, Cancel, Reset, Merge, and Actions. Below this is a 'Primary Information' section containing fields for CUSTOM FORM (Standard Case Form), PROFILE (Parent Company), STATUS (In Progress), NUMBER (72), ASSIGNED TO (26 Wolfe, M), PRIORITY (Medium), SUBJECT (Deliveries), COMPANY (16 Altima Technology), CONTACT (16 Altima Technology : Jeff Riggs), EMAIL(S) (jeffriggs@altima.com), SUBSIDIARY (Parent Company), PHONE, INCIDENT DATE (1/29/2015), PRODUCT, ORIGIN, INCIDENT TIME (10:30 pm), MODULE, INBOUND EMAIL ADDRESS, ITEM, TYPE, SERIAL/LOT NUMBER, CASE ISSUE, INBOUND MEMO, and CURRENT ASSIGNED TO (26). A note at the bottom left says 'Click here to create a new document signature request record.' The 'SampleTab' tab is circled in red.

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

## Create a Suitelet Email Form

The following sample shows how to create a Suitelet form to email the results.

### Script:

```
/*
 * A simple Suitelet for building an email form and sending out an email
 * from the current user to the recipient email address specified on the form.
 */
function simpleEmailForm(request, response)
{
    if (request.getMethod() == 'GET')
```

```

{
    var form = nlapiCreateForm('Email Form');
    var subject = form.addField('subject','text', 'Subject');
    subject.setLayoutType('normal','startcol')
    subject.setMandatory( true );
    var recipient = form.addField('recipient','email', 'Recipient email');
    recipient.setMandatory( true );
    var message = form.addField('message','textarea', 'Message');
    message.setDisplaySize( 60, 10 );
    form.addSubmitButton('Send Email');
    response.writePage(form);
}
else
{
    var currentuser = nlapiGetUser();
    var subject = request.getParameter('subject')
    var recipient = request.getParameter('recipient')
    var message = request.getParameter('message')
    nlapiSendEmail(currentuser, recipient, subject, message);
}
}

```

## Create a Form with a URL Field

This example shows how to add a URL field to a basic form. In this example, when the URL is clicked the user will be redirected to the New Employee record.

Several methods of [nlobjField](#) are used to change the displayed field to the correct parameters. The address called is built from the system address plus the address of the tasklink (in this case the New Employee form) which is retrieved using [nlapiResolveURL\(type, identifier, id, displayMode\)](#).

 **Note:** You need to specify 'https://' if the destination link is accessible via an authenticated NetSuite session.

### Script:

```

function SimpleFormWithUrl(request, response)
{
    if ( request.getMethod() == 'GET' )
    {
        var form = nlapiCreateForm('Simple Form');
        var field = form.addField('textfield','text', 'Text');
        field.setLayoutType('normal', 'startcol');
        form.addField('datefield','date', 'Date');
        form.addField('currencyfield','currency', 'Currency');
        form.addField('textareafield','textarea', 'Textarea');

        form.addField("enterempslink", "url", "", null, "enteremps" ).setDisplayType( "inline"
).setLinkText("Click Here to Enter Employee Records").setDefaultValue( "https://system.netsuite.com" + nlapiResolveURL( 'tasklink', 'EDIT_EMPLOYEE' ) );
    }
}

```

```

        form.addSubmitButton('Submit');

        response.writePage( form );
    }
else
    dumpResponse(request,response);
}

```

The screenshot shows a NetSuite form titled "Simple Form With URL". The form includes four input fields: "Text" (a single-line text input), "Date" (a date input with a calendar icon), "Currency" (a dropdown menu), and "Textarea" (a multi-line text area). Below the form, there is a link labeled "Click Here to Enter a new Employee". At the bottom of the form, there is a "Submit" button.

## Using Embedded Inline HTML in a Form

Here is an inline HTML example where the inline HTML is embedded in a NetSuite form. To embed inline HTML, add a field with the nlobjForm.addField method of the type 'inlinehtml', and use the nlobjField.setDefaultValue method to provide the HTML code.

### Script:

```

function SurveyInlineHTML(request,response)
{
    var form = nlapiCreateForm('Thank you for your interest in Wolfe Electronics', true);

    var htmlHeader = form.addField('custpage_header', 'inlinehtml').setLayoutType('outsideabove'
    ', 'startrow');
    htmlHeader
        .setDefaultValue("<p style='font-size:20px'>We pride ourselves on providing the best
        services and customer satisfaction. Please take a moment to fill out our survey.</
        p><br><br>");
}

var htmlInstruct = form.addField('custpage_p1', 'inlinehtml').setLayoutType('outsideabove',
'startrow');
htmlInstruct

```

```

.setDefaultValue("<p style='font-size:14px'>When answering questions on a scale of 1 to
10,
1 = Greatly Unsatisfied and 10 = Greatly Satisfied.</p><br><br>");

form.addField('custpage_lblproductrating', 'inlinehtml')
.setLayoutType('normal', 'startrow')
.setDefaultValue("<p style='font-size:14px'>How would you rate your satisfaction with o
ur products?</p>");

form.addField('custpage_rdoproducing', 'radio', '1', 'p1').setLayoutType('startrow');
form.addField('custpage_rdoproducing', 'radio', '2', 'p2').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '3', 'p3').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '4', 'p4').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '5', 'p5').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '6', 'p6').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '7', 'p7').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '8', 'p8').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '9', 'p9').setLayoutType('midrow');
form.addField('custpage_rdoproducing', 'radio', '10', 'p10').setLayoutType('endrow');

form.addField('custpage_lbservicerating', 'inlinehtml')
.setLayoutType('normal', 'startrow')
.setDefaultValue("<p style='font-size:14px'>How would you rate your satisfaction with o
ur services?</p>");

form.addField('custpage_rdservicerating', 'radio', '1', 'p1').setLayoutType('startrow');
form.addField('custpage_rdservicerating', 'radio', '2', 'p2').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '3', 'p3').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '4', 'p4').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '5', 'p5').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '6', 'p6').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '7', 'p7').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '8', 'p8').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '9', 'p9').setLayoutType('midrow');
form.addField('custpage_rdservicerating', 'radio', '10', 'p10').setLayoutType('endrow');

form.addSubmitButton('Submit');
form.addResetButton('Reset');

response.writePage(form);
}

```

### Thank you for your interest in Wolfe Electronics

[Submit](#) [Reset](#)

We pride ourselves on providing the best services and customer satisfaction. Please take a moment to fill out our survey.

When answering questions on a scale of 1 to 10, 1 = Greatly Unsatisfied and 10 = Greatly Satisfied.

How would you rate your satisfaction with our products?  
 1  2  3  4  5  6  7  8  9  10

How would you rate your satisfaction with our services?  
 1  2  3  4  5  6  7  8  9  10



# RESTlets

You can deploy server-side scripts that interact with NetSuite data following RESTful principles. RESTlets extend the SuiteScript API to allow custom integrations with NetSuite. Some benefits of using RESTlets include the ability to:

- Find opportunities to enhance usability and performance, by implementing a RESTful integration that is more lightweight and flexible than SOAP-based web services.
- Support stateless communication between client and server.
- Control client and server implementation.
- Use built-in authentication based on token or user credentials in the HTTP header.
- Develop mobile clients on platforms such as iPhone and Android.
- Integrate external Web-based applications such as Gmail or Google Apps.
- Create backends for Suitelet-based user interfaces.

RESTlets offer ease of adoption for developers familiar with SuiteScript and support more behaviors than NetSuite's SOAP-based web services, which are limited to those defined as SuiteTalk operations. RESTlets are also more secure than Suitelets, which are made available to users without login. For a more detailed comparison, see [RESTlets vs. Other NetSuite Integration Options](#).

For more information about the REST architectural style, a description of the constraints and principles is available at [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer).

For details about working with RESTlets, see:

- [Working with RESTlets](#)
- [RESTlets vs. Other NetSuite Integration Options](#)
- [Creating a RESTlet](#)
- [Debugging a RESTlet](#)
- [Sample RESTlet Code](#)
- [Sample RESTlet Input Formats](#)
- [RESTlet Status Codes and Error Message Formats](#)
- [Tracking and Managing RESTlet Activity](#)



**Important:** To maintain the highest security standards for connectivity, NetSuite periodically updates server certificates and the trusted certificate store we use for https requests. We expect client applications that integrate with NetSuite to use an up-to-date certificate store, to ensure that integrations do not break due to certificates expiring or changing security standards. This issue typically does not affect modern browsers because these clients are maintained to use the latest certificates.

## Working with RESTlets

You can invoke a RESTlet via an HTTP request to a system-generated URL. RESTlets send and receive content in a request-response model using HTTP verbs, HTTP headers, HTTP status codes, URLs, and standard data formats.

RESTlets support the entire SuiteScript API and general SuiteScript features such as debugging. For general information about using SuiteScript, see [SuiteScript - The Basics](#).

The following topics provide details specific to the RESTlet type of script:

- [RESTlet Script Execution](#)
- [Authentication for RESTlets](#)
- [RESTlet URL and Domain](#)
- [Using the REST roles Service to Get User Accounts, Roles, and Domains](#)
- [Supported Input and Output Content Types for RESTlets](#)
- [Supported Functions for RESTlets](#)
- [RESTlet Governance and Session Management](#)
- [RESTlet Debugging](#)
- [RESTlet Error Handling](#)
- [RESTlet Security](#)

## RESTlet Script Execution

The following steps provide an overview of the RESTlet execution process:

1. A client initiates an HTTP request for a system-generated URL. This request can come from an external client or from a client hosted by NetSuite.
2. The sender of the HTTP request is authenticated either through request-level credentials passed by the NetSuite-specific method `NLAuth`, the NetSuite implementation of OAuth 1.0, or through a check for an existing NetSuite session.
  - For an externally hosted client using `NLAuth`, request-level credentials are required.
  - For an externally hosted client using OAuth 1.0, a token is required.
  - For a NetSuite-hosted client, the existing NetSuite session is reused.
3. See [Authentication for RESTlets](#).
4. The RESTlet script is invoked, providing access to the server SuiteScript API.
5. A string, or an object that has been deserialized according to a predefined specification, is passed in to the RESTlet. See [Supported Input and Output Content Types for RESTlets](#).
6. The RESTlet interrogates the string or object, and can perform any of the full range of SuiteScript actions.
7. The RESTlet returns results as a string or a serialized object. `nlobjResponse`



**Important:** The URL used to invoke a RESTlet varies according to whether the RESTlet client is externally hosted or hosted by NetSuite. See [RESTlet URL and Domain](#).

## Authentication for RESTlets

Authentication is required for RESTlets. All calls to RESTlets are processed synchronously and RESTlets support a high number of concurrent requests, so the same credentials can be reused.

The way to provide login credentials for a RESTlet varies according to whether the RESTlet is called from an external client or from a client hosted by NetSuite, such as a client SuiteScript.



**Important:** RESTlets must use rest URLs to connect to NetSuite. These URLs include a data center-specific domain, and the rest subdomain. NetSuite strongly recommends using the REST roles Service in your code to dynamically discover the correct URLs to use for RESTlets. See [Using the REST roles Service to Get User Accounts, Roles, and Domains](#).

- For a RESTlet called from an external client, you can use OAuth or the NetSuite-specific method **NLAUTH** in the HTTP Authorization header. OAuth uses token-based authentication (TBA) to access resources on behalf of a user, eliminating the need to share login credentials such as username and password. See [Using OAuth in the Authorization Header](#).  
NLAUTH passes in NetSuite login credentials such as company ID, user name, password, and role. See [Using NLAUTH in the Authorization Header](#).
- For a RESTlet called from a client hosted by NetSuite, you do not need to pass authentication information in the HTTP request. A check for all valid NetSuite session cookies occurs, and this existing session is reused.



**Important:** RESTlet authentication can use either the HTTP Authorization header or all session cookies, but not both. Please ensure that your script uses only one form of authentication.

## Setting up Token-based Authentication for a RESTlet integration

NetSuite supports token-based authentication (TBA) a robust, industry standard-based mechanism that increases the overall security of the system. This authentication mechanism enables client applications to use a token to access NetSuite through APIs, eliminating the need for RESTlets to store user credentials. A token is valid for one specific company, user entity, and role only.



**Note:** Web Services Only roles are only for access to NetSuite through web services. Roles with the Web Services Only restriction will not work with RESTlets.

For more information, see the help topic [Getting Started with Token-based Authentication](#).

When you use token-based authentication, password rotation policies in the account do not apply to tokens and password management is unnecessary for your RESTlets integrations. Token-based authentication allows integrations to comply with any authentication policy that is deployed in a NetSuite account for UI login, such as SAML Single Sign-on, Inbound Single Sign-on, or Two-Factor Authentication. To enable token-based authentication, see the help topic [Enabling the Token-based Authentication Feature](#).

You can create a token and assign it to a user by logging in to NetSuite as an administrator and generating token credentials manually. NetSuite users can also generate token for themselves. See the help topic [Token-based Authentication Permissions](#).

For code snippets and examples of signature creation and token-based authentication, see [SuiteAnswer 42171](#) and [SuiteAnswer 42019](#).

For information about calling a token endpoint to issue or revoke a token, see the help topic [TBA: Calling a Token Endpoint](#) in the Token-based Authentication section of the Help Center.

## Using OAuth in the Authorization Header

When calling a RESTlet, follow the OAuth 1.0 specification to generate a token. A description of the OAuth 1.0 protocol and signature validation is available at <https://tools.ietf.org/html/rfc5849>.



**Note:** Supported Signature methods are HMAC-SHA1 and HMAC-SHA256.

To obtain the parameter values required for OAuth, refer to the confirmation page for your NetSuite token-based application. See the help topic [Creating Applications for Token-based Authentication](#).

OAuth passes in the following parameters:

- **oauth\_signature** (required) - Credentials to verify the authenticity of the request, generated by calling your application. The Token Secret and Consumer Secret are constructed as a key to sign the request, using a supported signature method (HMAC-SHA1 or HMAC-SHA256).
- **oauth\_version** (optional) - Must be set to "1.0".
- **oauth\_nonce** (required) - Passes in a unique, random, alphanumeric string. String must be a minimum of 6 characters, and the maximum length is 64 characters. Used to verify that a request has never been made before.
- **oauth\_signature\_method** (required) - Must be set to HMAC-SHA1 or HMAC-SHA256. Declares which signature method is used.
- **oauth\_consumer\_key** (required) - Consumer Key (client application ID) generated for the token-based application in NetSuite. The unique value is matched to the token to establish ownership of the token.
- **oauth\_token** (required) - Token ID generated for the token-based application in NetSuite.
- **oauth\_timestamp** (required) - Passes in a positive integer expressed as the number of seconds since January 1, 1970 GMT.
- **realm** (required) - NetSuite company ID

To map error codes to a parameter issue, see [Notes about RESTlet Errors](#).



**Note:** NetSuite recommends entering the key value pairs without leading or trailing spaces (realm="000068" not realm= "000068" ) as shown in the following example.

## Example RESTlet request using OAuth to access a protected resource

```
GET https://rest.na1.netsuite.com/app/site/hosting/restlet.nl?script=1&deploy=1 HTTP/1.1
Authorization:
    OAuth oauth_signature="MgN1gZztYspNQXA576pIPD14OWM%3D",
    oauth_version="1.0",
    oauth_nonce="207310548",
    oauth_signature_method="HMAC-SHA1",
    oauth_consumer_key="fvFwnmvurChjol7SZiF2pQ1oJ%2FceRV8vqA%2FrZtzLEo%3D",
    oauth_token="00076e1415667a6c555f5d43582134c87d6367ab456fd2",
    oauth_timestamp="1418647040",
    realm="000068"

HTTP/1.1 200 OK
Date: Mon, 15 Dec 2014 12:37:42 GMT
Content-Type: text/html
Content-Length: 12

Hello World!
```



## Using NLAuth in the Authorization Header

NLAuth passes in the following login credentials:

- nlauth\_account (required) - NetSuite company ID
- nlauth\_email (required) - NetSuite user name
- nlauth\_signature (required) - NetSuite password
- nlauth\_role (optional) - internal ID of the role used to log in to NetSuite

 **Note:** If a user has a default role defined, this role can be used for login when the role parameter is not passed in the authorization header.

 **Important:** NetSuite RESTlet authentication only accepts special characters that are URL encoded. If your credentials contain special characters, replace each special character with its appropriate URL encoding. For additional information on URL encoding, see [http://www.w3schools.com/tags/ref\\_urlencode.asp](http://www.w3schools.com/tags/ref_urlencode.asp)

## NLAuth Authorization Header Formatting

For NLAuth, the Authorization header should be formatted as:

```
NLAuth<space><comma-separated parameters>.
```

 **Note:** NetSuite recommends entering the key value pairs without leading or trailing spaces (nlauth\_account=123456, not nlauth\_account= 123456 ,) as shown in the following example.

For example:

```
Authorization: NLAuth nlauth_account=123456, nlauth_email=jsmith@ABC.com, nlauth_signature=xxx
xxxx, nlauth_role=41
```

 **Important:** NetSuite provides a REST roles service that you can use to determine a user's account and role. This service makes it possible to support RESTlet login when account and role are unknown, for example, in mobile applications. See [Using the REST roles Service to Get User Accounts, Roles, and Domains](#).

## RESTlet URL and Domain

The URL used for a RESTlet HTTP request varies according to whether the RESTlet is called from an external client or from a client hosted by NetSuite.

- For a RESTlet called from an external client, the URL must be an absolute URL. This URL includes the RESTlet domain specific to the data center that houses your NetSuite account data. This domain might be <https://rest.netsuite.com>, <https://rest.na1.netsuite.com>, or a similar variant. In an integration, you **must** dynamically discover this domain. If you use an incorrect domain, the request fails. You can use the REST roles service to dynamically discover the correct domain. For details, see [Using the REST roles Service to Get User Accounts, Roles, and Domains](#).
- For a RESTlet called from a client hosted by NetSuite, the URL should be a relative URL. That is, the URL does not include the domain.

When the NetSuite account is hosted at system.netsuite.com, the RESTlet also uses the same base URL. For example, if you use /app/site/hosting/restlet.nl (as seen in the following screenshot) and deploy that RESTlet in accounts located in different data centers, the application will correctly prefix /app/site/hosting/restlet.nl with the base URL, as appropriate for the location of each account.

The following RESTlet deployment record shows examples of URLs.

- The URL field displays the relative URL, used by NetSuite-hosted clients for any references made to objects inside NetSuite.
- The External URL field displays an absolute URL that could be used to call the RESTlet from outside of NetSuite. However, any integrations you create must dynamically discover the domain that makes up the first part of this URL (the parts that differ from the text shown under the URL field).

The screenshot shows a 'Script Deployment' page with the following details:

SCRIPT	STATUS
SuiteSocial News Feed RESTlet	Released
TITLE	LOG LEVEL
SuiteSocial News Feed RESTlet	Debug
ID	URL
customdeploy_newsfeed_restlet	/app/site/hosting/restlet.nl?script=218&deploy=1
DEPLOYED	EXTERNAL URL
<input checked="" type="checkbox"/>	<a href="https://rest.netsuite.com/app/site/hosting/restlet.nl?script=218&amp;deploy=1">https://rest.netsuite.com/app/site/hosting/restlet.nl?script=218&amp;deploy=1</a>

**i Note:** RESTlets use the same debugging domain as other SuiteScript types, <https://debugger.netsuite.com>. Whether the RESTlet client is hosted externally or by NetSuite does not change the debugger domain used. See [RESTlet Debugging](#).

## Using the REST roles Service to Get User Accounts, Roles, and Domains

NetSuite provides a REST roles service that returns the following data when you submit a user email address and password:

- account(s) and role(s) available to the user
- REST, web services, and general system domains to be used for external client access to NetSuite

**i Note:** RESTlets are part of SuiteScript. They are **not** part of NetSuite's web services feature. Be aware that if a role has the Web Services Only option set to true, a user logged in through that role is permitted to send web services calls only. RESTlet calls receive an INVALID\_LOGIN\_CREDENTIALS error response.

The roles service fills the following needs:

- Support for RESTlet login when the user's account and role are unknown.
- Dynamic discovery of correct URLs for external client access to each NetSuite account. Discovery of the following domains is supported:
  - restDomain - for example: <https://rest.netsuite.com>, <https://rest.na1.netsuite.com>, and similar variants
  - systemDomain - for example: <https://system.netsuite.com>, <https://system.na1.netsuite.com>, and similar variants

- webservicesDomain - for example: <https://webservices.netsuite.com>, <https://webservices.na1.netsuite.com>, and similar variants

This discovery is required to support the hosting of customer accounts in multiple data centers. Each data center has a different domain, so the domain to be used for external client access depends upon the data center hosting each NetSuite account.



**Important:** NetSuite hosts customer accounts in multiple data centers. For that reason, the correct domain for external client RESTlet access varies depending on the data center hosting the account. Your integration should **must** incorporate logic that dynamically determines the correct RESTlet domain. The REST roles service is designed for this purpose. For details on using this service, see [Using the REST roles Service to Get User Accounts, Roles, and Domains](#).

Additionally, if you have a web services integration that uses the 2012.1 or an earlier WSDL, you must use the roles service to discover the web services URL for your account. For sample web services code that calls the REST roles service, see the help topics [Java REST Example for Web Services Domain Discovery](#) and [.NET REST Example for Web Services Domain Discovery](#). If you use the 2012.2 endpoint or later, you can use a web services operation, `getDataCenterUrls`, to obtain the appropriate web services URL.

## URLs for Sandbox, Sandbox 2.0, and Release Preview Environments

For information on the correct URL to use for Sandbox, Sandbox 2.0, and Release Preview Environments for RESTlet clients, see the help topic [Understanding NetSuite URLs and Data Centers](#).

## Sample REST roles Request

To get the available accounts, roles, and domains for a user, submit the user's email address and password in the authorization header, as shown in the following example:

```
URL: https://rest.netsuite.com/rest/roles

Headers:
GET /rest/roles HTTP/1.1
Accept: */*
Accept-Language: en-us
Authorization: NLAUTH nlauth_email=johnsmith@xxxxx.com, nlauth_signature=****
```

## Sample REST roles Response

The roles service returns a list of account(s) available to the user, and for each account, the associated roles and domains, as shown in the following example:

```
[{"account":{"internalId":"1234567","name":"Test Account1"},"role":{"internalId":3,"name":"Administrator"}, "dataCenterURLs":{"restDomain":"https://rest.netsuite.com","systemDomain":"https://system.netsuite.com", "webservicesDomain":"https://webservices.netsuite.com"}}, {"account":{"internalId":"1234678","name":"Test Account2"}, "role":{"internalId":3,"name":"Administrator"}, "dataCenterURLs":{"restDomain":"https://rest.netsuite.com","systemDomain":"https://system.netsuite.com", "webservicesDomain":"https://webservices.netsuite.com"}},
```

```
{"account":{"internalId":"1234789","name":"Test Account3","role":{"internalId":3,"name":"Administrator"},"dataCenterURLs":{"restDomain":"https://rest.netsuite.com","systemDomain":"https://system.netsuite.com","webservicesDomain":"https://webservices.netsuite.com"}}]
```

The role to use for login can be selected from this list. And you can use this list to determine the domains to specify in RESTlet and web services requests.

## Supported Input and Output Content Types for RESTlets

RESTlets support JSON and plain text content types for input and output. For each RESTlet, output content type is the same as input content type.

You must set the content type in the HTTP Content-Type header. You can use the following values to specify the input/output content type for a RESTlet:

- application/json
- text/plain

If you specify a content type other than JSON or text, a 415 error is returned with the following message:

Invalid content type. You can only use application/json or text/plain with RESTlets.

## Using JSON Objects and Arrays

JSON is an acronym for JavaScript Object Notation, which is a subset of JavaScript. This special object notational construct is a syntax used to pass JavaScript objects containing name/value pairs, arrays, or other objects. The following JSON formatting is used for RESTlets:

- Each JSON object is an unordered set of name/value pairs, or members, enclosed in curly braces.
  - Each member is followed by a comma, which is called a value separator.
  - Within each member, the name is separated from the value by a colon, which is called a name separator.
  - Each name and each value is enclosed in quotation marks.
- Each JSON array is an ordered sequence of values, enclosed in square braces. Array values are separated by commas.

For examples of how to format JSON input for restlets, see [Sample RESTlet Input Formats](#).

## Supported Functions for RESTlets

RESTlets currently support a subset of HTTP methods, as shown in the following table:

HTTP Method	Input	Output	Description
GET	Parameter Object	Object	Requests a representation of the specified resource.
POST	Object	Object	Submits data to be processed, for example from an HTML form. Data is included in the body of the request, and can result in



HTTP Method	Input	Output	Description
			creation of a new resource, updates of existing resource(s), or both.
DELETE	Parameter Object	No Content	Passes in the ID and record type of the resource to be deleted, so that nlapiDeleteRecord or other delete-related logic can be called. This method does not return anything.
PUT	Object	Object	Uploads a representation of the specified resource.

The functions that implement these methods are specified on a RESTlet's script record. Each RESTlet must have a function for at least one of these HTTP methods. Each HTTP method can call any SuiteScript nlapi functions. See [Create the RESTlet Script Record](#).

For examples of these functions in RESTlets, see [Sample RESTlet Code](#).

## RESTlet Governance and Session Management

The SuiteScript governance model tracks usage units on two levels: API level and script level. At the API level, RESTlets have the same usage limits as other types of SuiteScripts. At the script level, RESTlets allow 5,000 usage units per script, a limit five times greater than Suitelets and most other types of SuiteScripts. For more information, see [SuiteScript Governance](#).

There is a limit of 10MB per string used as RESTlet input or output.

SuiteScript currently does not support a logout operation similar to the one used to terminate a session in SuiteTalk.

## RESTlet Debugging

You can use the SuiteScript Debugger to debug RESTlet scripts, in the same manner that you use it to debug other types of server SuiteScripts. RESTlets use the same debugging domain as other SuiteScript types, <https://debugger.netsuite.com>. Both ad-hoc debugging and deployed debugging are supported for RESTlets. For general instructions for using the Debugger, see [Working with the SuiteScript Debugger](#).



**Important:** For deployed debugging of a RESTlet, you need to set the session cookies of the client application that run the RESTlet to the same cookies listed for the RESTlet in the Debugger. Also, you must remove the authorization header from your RESTlet before debugging. For more details, see [Debugging a RESTlet](#).

## RESTlet Error Handling

RESTlets return standard HTTP status codes for their contained HTTP requests. A standard success code is returned for a successful request. Standard error codes are returned for errors due to unparsable input, authentication failure, lack of server response, use of an unsupported method, and use of an invalid content type or data format for input. In most cases, generic HTTP error messages are returned. The format used for error messages is the same as the specified format for input: JSON or plain text. For more details, see [RESTlet Status Codes and Error Message Formats](#).

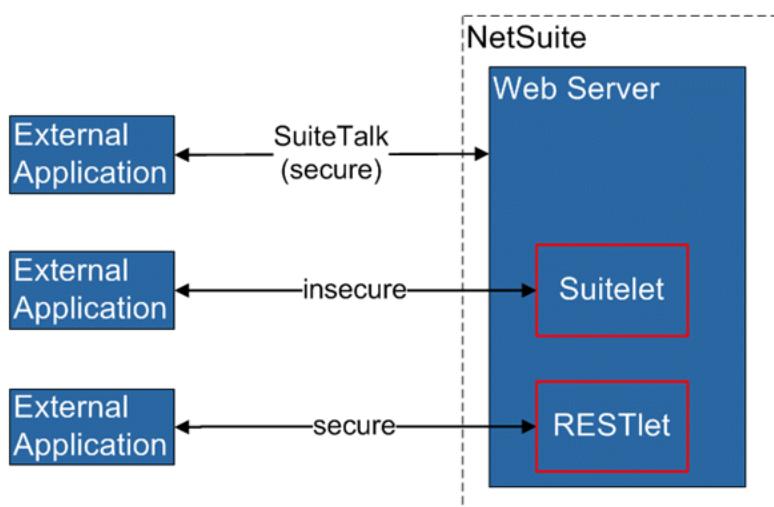
RESTlets also support the SuiteScript nlapiCreateError function. You can include this API in your code to abort script execution when an error occurs. For details, see [Error Handling APIs](#).

## RESTlet Security

The URLs for RESTlet HTTP requests can vary, but all resources are protected by TLS encryption. NetSuite supports TLS 1.0, 1.1, and 1.2 encryption for forms.netsuite.com, system.netsuite.com, and other NetSuite domains. Only requests sent using TLS encryption are granted access. For more on RESTlet domains, see [RESTlet URL and Domain](#).

## RESTlets vs. Other NetSuite Integration Options

RESTlets provide one option for integration with NetSuite. Other options include SOAP-based web services through SuiteTalk, and Suitelets.



Review the following for comparisons of these integration options:

- RESTlets Compared to SuiteTalk
- RESTlets Compared to Suitelets

## RESTlets Compared to SuiteTalk

The following table compares the characteristics of RESTlets with those of SuiteTalk's SOAP-based web services:

Attribute	RESTlets	SuiteTalk
Supported Operations	get, search, add, update (heterogeneous)	get, search, add, update (homogenous)
Authentication Supported?	Yes	Yes

Attribute	RESTlets	SuiteTalk
Supported HTTP Methods	GET, PUT, POST, DELETE	POST
Passing of Login Details	in authorization header	in body (SOAP)
Passing of Parameters	GET parameters on URL	all parameters in body (SOAP)
Supported Content Types	JSON, text/xml (explicit)	text/xml (explicit)
Environment	lightweight, more suitable for mobile devices, bundleable	heavy programming and deployment environment (C#, Java)
URL Clarity?	<p>Yes</p> <p><a href="https://rest.netsuite.com/app/site/hosting/restlet.nl?script=57&amp;deploy=1&amp;recordtype=salesorder&amp;id=21480">https://rest.netsuite.com/app/site/hosting/restlet.nl?script=57&amp;deploy=1&amp;recordtype=salesorder&amp;id=21480</a></p> <p>(Note that for clients hosted by NetSuite, use the relative URL that does not include the domain.)</p>	<p>No</p> <p><a href="https://webservices.netsuite.com/services/NetSuitePort_2011_1">https://webservices.netsuite.com/services/NetSuitePort_2011_1</a></p>

 **Note:** SuiteTalk is recommended for system-to-system integrations.

## RESTlets Compared to Suitelets

The following table compares the characteristics of RESTlets with those of Suitelets:

Attribute	RESTlets	Suitelets
Supported Operations	get, search, add, update	get, search, add, update
Authentication Supported?	Yes	No , when available without login and executed as admin programmatically Yes, when accessed from a browser by a logged-in NetSuite user
Script Functions and HTTP Methods	individual script function for each HTTP method	one script function for all HTTP method
Content Handling	built-in handling of JSON input/output	must write code to convert JSON input/output
Governance	5,000 usage units per script	1,000 usage units per script
URL Clarity?	<p>Yes</p> <p><a href="https://rest.netsuite.com/app/site/hosting/restlet.nl?script=57&amp;deploy=1&amp;recordtype=salesorder&amp;id=21480">https://rest.netsuite.com/app/site/hosting/restlet.nl?script=57&amp;deploy=1&amp;recordtype=salesorder&amp;id=21480</a></p>	<p>No</p> <p><a href="https://forms.netsuite.com/app/site/hosting/scriptlet.nl?script=62&amp;deploy=1&amp;compid=824056&amp;h=ec041b59b3075bec783d">https://forms.netsuite.com/app/site/hosting/scriptlet.nl?script=62&amp;deploy=1&amp;compid=824056&amp;h=ec041b59b3075bec783d</a></p>

Attribute	RESTlets	Suitelets
	(Note that for clients hosted by NetSuite, use the relative URL that does not include the domain.)	

## Creating a RESTlet

To run a RESTlet in NetSuite, you must first define your client code and behavior, then define your RESTlet and its required functions. The client will send requests to the RESTlet URL generated by NetSuite.

To define a RESTlet:

1. Create a JavaScript file for your RESTlet code.
2. Load the file into NetSuite.
3. Create a script record where you define SuiteScript functions for one or more HTTP methods.
4. Define all runtime options on the Script Deployment page.

See the following for instructions for these tasks:

- [Create the RESTlet File and Add It to the File Cabinet](#)
- [Create the RESTlet Script Record](#)
- [Define RESTlet Deployment\(s\)](#)

### Create the RESTlet File and Add It to the File Cabinet

1. Create a .js file and add your code to it, in the same manner that you create other types of SuiteScript files, as described in [Step 1: Create Your Script](#).  
This single script file should include GET, POST, DELETE, or PUT function(s) as necessary.
2. After you have created a .js file with your RESTlet code, you need to add this file to the NetSuite file cabinet.

The following steps describe how to add the file manually. If you are using SuiteCloud IDE, this process is automated. For more information, see [Step 2: Add Script to NetSuite File Cabinet](#).

1. Go to Documents > Files > File Cabinet, and select the folder where you want to add the file.  
It is recommended that you add your file to the SuiteScripts folder, but it can be added to any other folder of your choice.
2. Click **Add File**, and browse to the .js file.

### Create the RESTlet Script Record

After you have added a RESTlet file to the file cabinet, you can create a NetSuite script record.

#### To create a RESTlet script record:

1. Go to Setup > Customization > Scripts > New, and click **RESTlet**.

2. Complete fields in the script record and save.

Although you do not need to set every field on the Script record, at a minimum you must provide a **Name** for the Script record, load your SuiteScript file to the record, and specify at least one of the following executing functions on the Scripts tab: GET, POST, DELETE, or PUT.

You can specify more than one of these functions as desired. These functions should all be in the main script file. If these functions call functions in other script files, you need to list those files as library script files.

For more details about creating a script record, see [Steps for Creating a Script Record](#).

## Define RESTlet Deployment(s)

After you have created a RESTlet script record, you need to define at least one deployment. For details about defining script deployments, see [Step 5: Define Script Deployment](#) and [Steps for Defining a Script Deployment](#)

You can define multiple deployments per RESTlet.

### To define a RESTlet script deployment.

1. Do one of the following:
  - When you save your Script record, you can immediately create a Script Deployment record by selecting **Save and Deploy** from the Script record **Save** button.
  - If you clicked **Save**, immediately afterwards you can click **Deploy Script** on the script record.
  - If you want to update a deployment that already exists, go to Customization > Scripting > Script Deployments > [deployment] > Edit.

2. Complete fields in the script deployment record and click **Save**.

If you want to debug the script, set the **Status** to **Testing**.



**Note:** After you have saved a RESTlet deployment, the deployment record includes the URL used to invoke the RESTlet. For a RESTlet called from an externally hosted client, use the **External URL**. For a RESTlet called from a client hosted by NetSuite, use the **URL** that does not include the domain. See [RESTlet URL and Domain](#).

## Debugging a RESTlet

You can use the NetSuite Debugger to debug RESTlet code in the same manner that you debug other types of SuiteScript code, as described in the NetSuite Help Center topic [Debugging SuiteScript](#).

- To debug code snippets before you have a RESTlet script record that has been deployed, called ad-hoc debugging, follow the instructions in [Ad Hoc Debugging](#). (Be sure not to include the RESTlet's authorization header in the code snippets to be debugged, as this header can prevent the debugger from working.)
- To debug an existing script that has a defined deployment, called deployed debugging, follow the steps below.



**Important:** In addition to debugging RESTlet script code, it is recommended that you test the HTTP request to be sent to the RESTlet. Free tools are available for this purpose. See [RESTlet HTTP Testing Tools](#).

### To debug a deployed RESTlet:

1. Before you deploy a RESTlet to be debugged, ensure that the script does not include the HTTP authorization header, as this header can prevent the debugger from working.
2. Ensure that on the script deployment record, the **Status** value is set to **Testing**.
3. Go to Customization > Scripting > Script Debugger, or log in to the debugger domain <https://debugger.netsuite.com>. (See [Deployed Debugging](#) for details.)
4. Click the **Debug Existing** button in the main Script Debugger page.



**Note:** This button only appears when you have deployed scripts with the status is set to **Testing**.

5. Select the RESTlet script that you want to debug in the Script Debugger popup.  
After you click the **Select** option button, the RESTlet's cookies display in a banner.
6. Copy the cookies and paste them into a text file so that you have them available.
7. Click the **Select and Close** button in the Script Debugger popup.  
The main Script Debugger page displays a message that it is waiting for user action.
8. Set the cookies in your client application to the values you copied in step 5, and send the RESTlet request.  
The main Script Debugger page displays the script execution as pending at the NetSuite function `restletwrapper(request)`.
9. You have the following options for debugging your script code:

- Click the **Step Over** button to begin stepping through each line of code.
- Add watches and evaluate expressions.
- Set break points and click the **Run** button to run the code. The Debugger will stop code execution at the first break point set.
- Set no break points, click the **Run** button, and have the Debugger execute the entire piece of code.

See [SuiteScript Debugger Interface](#) for information on stepping into/out of functions, adding watches, setting and removing break points, and evaluating expressions.

## Debugging Timeout Errors

If a timeout error occurs during debugging, check for the following:

- Invalid or missing NetSuite version  
Ensure that you have correctly copied the session cookies, as described in the steps above. These cookies includes a valid NetSuite version.
- Incorrect domain such as <https://rest.netsuite.com>  
Ensure that you have logged in to <https://debugger.netsuite.com>.

## RESTlet HTTP Testing Tools

You can use the tools of your choosing to test the HTTP request to be sent to a RESTlet.

If you have installed and set up SuiteCloud IDE, a debug client is available for your use. The RESTlet/Suitelet Debug Client enables you to debug deployed RESTlet and Suitelet SuiteScripts with the SuiteCloud IDE Debugger. The client is only accessible after a debug session is started. For details about this client, see the help topic [Using the RESTlet/Suitelet Debug Client](#). For information about SuiteCloud IDE, see the help topic [Working with SuiteCloud IDE](#).

In addition, the following free tools are available:

- **Send HTTP Tool**  
This tool is a free HTTP Request builder that you can use to send an HTTP request to the RESTlet and analyze the response.  
<http://soft-net.net/SendHTTPTool.aspx>
- **Fiddler**  
This tool is a free Web debugging proxy that you can use to log HTTP traffic, and inspect the HTTP request and response for the RESTlet.  
<http://www.fiddler2.com/fiddler2/>



**Warning:** The above information about free tools is provided as a courtesy and is not intended as an endorsement or recommendation of these tools.

## Sample RESTlet Code

The following examples provide sample RESTlet code:

- Simple Example to Get Started
- Example Code Snippets of HTTP Methods
- Example RESTlet Called from a Portlet Script
- Example RESTlet Request from Android
- Example RESTlet Request Using nlapiRequestURL

## Simple Example to Get Started

Use the following example as a very basic GET method test when you are getting started with RESTlets:

```
function sayhi()
{
    var o = new Object();
    o.sayhi = 'Hello World! ';
    return o;
}
```

## Example Code Snippets of HTTP Methods

The following code snippets provide examples of RESTlet functions.

### GET Method

```
// Get a standard NetSuite record
function getRecord(datain)
{
    return nlapiLoadRecord(datain.recordtype, datain.id); // e.g recordtype="customer", id=769
}
```

Query parameters:

```
recordtype=customer&id=769
```

### POST Method

```
// Create a standard NetSuite record
function createRecord(datain)
{
    var err = new Object();

    // Validate if mandatory record type is set in the request
    if (!datain.recordtype)
    {
        err.status = "failed";
```



```

        err.message= "missing recordtype";
        return err;
    }

    var record = nlapiCreateRecord(datain.recordtype);

    for (var fieldname in datain)
    {
        if (datain.hasOwnProperty(fieldname))
        {
            if (fieldname != 'recordtype' && fieldname != 'id')
            {
                var value = datain[fieldname];
                if (value && typeof value != 'object') // ignore other type of parameters
                {
                    record.setFieldValue(fieldname, value);
                }
            }
        }
    }

    var recordId = nlapiSubmitRecord(record);
    nlapiLogExecution('DEBUG','id='+recordId);

    var nlobj = nlapiLoadRecord(datain.recordtype,recordId);
    return nlobj;
}

```

Request Payload:

```
{"recordtype":"customer", "entityid":" John Doe", "companyname":"ABCTools Inc", "subsidiary":"1", "email":"jdoe@email.com"}
```

## DELETE Method

```
// Delete a standard NetSuite record
function deleteRecord(datain)
{
    nlapiDeleteRecord(datain.recordtype, datain.id); // e.g recordtype="customer", id="769"
}
```

Query parameters:

```
recordtype=customer&id=769
```

## Example RESTlet Called from a Portlet Script

- Portlet Script Code - Calls RESTlet and Sets Search Criteria
- RESTlet Code - Gets Data based on Portlet Script Criteria



## Portlet Script Code - Calls RESTlet and Sets Search Criteria

```

function getAccount () { return '1234567'; }

function getRESTletURL()
{
    return 'https://rest.netsuite.com/app/site/hosting/restlet.nl?script=27&deploy=1&c=' + getAccount(); // for phasing
}

function credentials()
{
    this.email='jsmith@abcauto.com';
    this.account=getAccount();
    this.role='3';
    this.password='mysecretpwd';
}

// Portlet function
function displayOpportunities(portlet)
{
    portlet.setTitle('Opportunities');
    var col = portlet.addColumn('id','text', 'ID', 'LEFT');
    col.setURL(nlapiResolveURL('RECORD','opportunity'));
    col.addParamToURL('id','id', true);
    portlet.addColumn('title','text', 'Opportunity', 'LEFT');
    portlet.addColumn('customer','text', 'Customer', 'LEFT');
    portlet.addColumn('salesrep','text', 'Sales Rep', 'LEFT');
    portlet.addColumn('amount','currency', 'Amount', 'RIGHT');
    portlet.addColumn('probability','currency', 'Probability', 'RIGHT');

    var opps = getOpportunities();
    if ( opps != null && opps.length > 0 )
    {
        for ( var i=0; i < opps.length ; i++ )
        {
            portlet.addRow(opps[i]);
        }
    }
}

function getOpportunities()
{
    var url = getRESTletURL() + '&daterange=daysAgo90&&probability=10';
    var cred = new credentials();

    var headers = new Array();
    headers['User-Agent-x'] = 'SuiteScript-Call';
    headers['Authorization'] = 'NLAAuth nlauth_account=' + cred.account + ', nlauth_email=' + cred.email + ', nlauth_signature=' + cred.password + ', nlauth_role=' + cred.role;
    headers['Content-Type'] = 'application/json';

    var response = nlapiRequestURL( url, null, headers );
}

```



```

var responsebody = JSON.parse(response.getBody());

var error = responsebody['error'];
if (error)
{
    var code = error.code;
    var message = error.message;
    nlapiLogExecution('DEBUG', 'failed: code=' + code +'; message=' + message);
    nlapiCreateError(code, message, false);
}

return responsebody['nssearchresult'];
}

function opportunity(internalid, title, probability, amount, customer, salesrep)
{
    this.id = internalid;
    this.title = title;
    this.probability = probability;
    this.amount = amount;
    this.customer = customer;
    this.salesrep = salesrep;
}

```

## RESTlet Code - Gets Data based on Portlet Script Criteria

```

function opportunity(internalid, title, probability, amount, customer, salesrep)
{
    this.id = internalid;
    this.title = title;
    this.probability = probability;
    this.amount = amount;
    this.customer = customer;
    this.salesrep = salesrep;
}

// RESTlet Get NetSuite record data
function getRecords(datain)
{
    var filters = new Array();
    var daterange = 'daysAgo90';
    var projectedamount = 0;
    var probability = 0;
    if (datain.daterange) {
        daterange = datain.daterange;
    }
    if (datain.projectedamount) {
        projectedamount = datain.projectedamount;
    }
    if (datain.probability) {
        probability = datain.probability;
    }
}

```

```

filters[0] = new nlobjSearchFilter( 'trandate', null, 'onOrAfter', daterange ); // like day
sAgo90
filters[1] = new nlobjSearchFilter( 'projectedamount', null, 'greaterthanorequalto', projec-
tedamount);
filters[2] = new nlobjSearchFilter( 'probability', null, 'greaterthanorequalto', probabilit-
y );

// 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' );
columns[7] = new nlobjSearchColumn( 'title' );

// Execute the search and return results

var opps = new Array();
var searchresults = nlapiSearchRecord( 'opportunity', null, filters, columns );

// Loop through all search results. When the results are returned, use methods
// on the nlobjSearchResult object to get values for specific fields.
for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
{
    var searchresult = searchresults[ i ];
    var record = searchresult.getId( );
    var salesrep = searchresult.getValue( 'salesrep' );
    var salesrep_display = searchresult.getText( 'salesrep' );
    var salesrep_email = searchresult.getValue( 'email', 'salesrep' );
    var customer = searchresult.getValue( 'entity' );
    var customer_display = searchresult.getText( 'entity' );
    var customer_email = searchresult.getValue( 'email', 'customer' );
    var expectedclose = searchresult.getValue( 'expectedclosedate' );
    var projectedamount = searchresult.getValue( 'projectedamount' );
    var probability = searchresult.getValue( 'probability' );
    var title = searchresult.getValue( 'title' );

    opps[opps.length++] = new opportunity( record,
        title,
        probability,
        projectedamount,
        customer_display,
        salesrep_display);
}

var returnme = new Object();
returnme.nssearchresult = opps;
return returnme;
}

```

## Example RESTlet Request from Android

```

HttpPost post = new HttpPost( URL + urlParams );

HttpParams httpParameters = new BasicHttpParams();
HttpConnectionParams.setConnectionTimeout( httpParameters, 20000 );
HttpConnectionParams.setSoTimeout( httpParameters, 42000 );

String authorization = "NLAuth nlauth_account=" + account + ", nlauth_email=" + email + ", nlau
th_signature=" + password + ", nlauth_role=" + role + "";
post.setHeader( "Authorization", authorization );
post.setHeader( "Content-Type", "application/json" );
post.setHeader( "Accept", "*/*" );

post.setEntity( new StringEntity( "{\"name\":\"John\"}" /*input data*/ ) );

HttpClient client = new DefaultHttpClient( httpParameters );
BufferedReader in = null;

HttpResponse response = client.execute( post );

in = new BufferedReader( new InputStreamReader( response.getEntity().getContent() ) );
StringBuffer sb = new StringBuffer( "" );
String line;
String NL = System.getProperty( "line.separator" );
while ( (line = in.readLine()) != null )
{
    sb.append( line + NL );
}
in.close();
String result = sb.toString();

```

## Example RESTlet Request Using nlapiRequestURL

```

function credentials(){
    this.email = "msmith@email.com";
    this.account = "1234567";
    this.role = "3";
    this.password = "*****";
}

function replacer(key, value){
    if (typeof value == "number" && !isFinite(value)){
        return String(value);
    }
    return value;
}

//Setting up URL
var url = "https://rest.netsuite.com/app/site/hosting/restlet.nl?script=260&deploy=1";

//Calling credential function

```

```

var cred = new credentials();

//Setting up Headers
var headers = {"User-Agent-x": "SuiteScript-Call",
    "Authorization": "NLAuth nlauth_account=" + cred.account + ", nlauth_email=" + cred.email +
        ", nlauth_signature= " + cred.password + ", nlauth_role=" + cred.role,
    "Content-Type": "application/json"};

//Setting up Datainput
var jsonobj = {"recordtype": "customer",
    "entityid": "John Doe",
    "companyname": "ABC Company",
    "subsidiary": "1",
    "email": "jdoe@email.com"}

//Stringifying JSON
var myJSONText = JSON.stringify(jsonobj, replacer);

var response = nlapiRequestURL(url, myJSONText, headers);

//Below is being used to put a breakpoint in the debugger
var i=0;

//*****RESTLET Code*****

// Create a standard NetSuite record
function createRecord(datain)
{
    var err = new Object();

    // Validate if mandatory record type is set in the request
    if (!datain.recordtype)
    {
        err.status = "failed";
        err.message = "missing recordtype";
        return err;
    }

    var record = nlapiCreateRecord(datain.recordtype);

    for (var fieldname in datain)
    {
        if (datain.hasOwnProperty(fieldname))
        {
            if (fieldname != 'recordtype' && fieldname != 'id')
            {
                var value = datain[fieldname];
                if (value && typeof value != 'object') // ignore other type of parameters
                {
                    record.setFieldValue(fieldname, value);
                }
            }
        }
    }
}

```

```

        }
    }

var recordId = nlapiSubmitRecord(record);
nlapiLogExecution('DEBUG','id='+recordId);

var nlobj = nlapiLoadRecord(datain.recordtype,recordId);
return nlobj;
}

```

## Sample RESTlet Input Formats

The following examples illustrate how to format input for RESTlets for the JSON content type:

- Customer Record Format
- Item Record Format
- Item Pricing Formats
- Sales Order Record Format

For a general explanation of JSON, see [Using JSON Objects and Arrays](#).

### Customer Record Format

#### JSON

```
{
    "shipcomplete":false,
    "giveaccess":false,
    "globalsubscriptionstatus":"1",
    "isperson":false,
    ... more body fields...
    "consoldepositbalance":0.00,
    "entityid":"John Doe",
    "addressbook":
    [
        {"zip":"94404","phone":"650-627-1000"},
        {"zip":"94403","phone":"650-627-1001"}
    ],
    "consoloverduebalance":0.00,
    "overduebalance":0.00,
    "creditholdoverride":"AUTO",
    "resubscribelink":"Send Subscription Email"
}
```

#### XML

```
<shipcomplete>F</shipcomplete>
<globalsubscriptionstatus>1</globalsubscriptionstatus>
<giveaccess>F</giveaccess>
<isperson>F</isperson>
```



```

<datecreated>12/19/2010 10:26 pm</datecreated>
<salesrep>-5</salesrep><currency>1</currency>
<lastmodifieddate>12/20/2010 10:14 am</lastmodifieddate>
<id>1185</id>
<emailtransactions>F</emailtransactions>
<balance>0.00</balance>
<entitystatus>13</entitystatus>
<isbudgetapproved>F</isbudgetapproved>

... more fields...

<entityid>John Doe</entityid>
<isinactive>F</isinactive>
<addressbook>
  <zip>94404</zip>
  <phone>650-627-1000</phone>
  <defaultshipping>T</defaultshipping>
  <addrtext>Netsuite100 Mission StreetFoster City CA 94404United States</addrtext>
  <state>CA</state>
  <addressee>Netsuite</addressee>
  <isresidential>F</isresidential>
  <label>Home</label>
  <city>Foster City</city>
  <country>US</country>
  <displaystate>California</displaystate>
  <dropdownstate>CA</dropdownstate>
  <addr1>100 Mission Street</addr1>
  <override>F</override>
  <defaultbilling>T</defaultbilling>
</addressbook>
<addressbook>
  <zip>94403</zip>
  <phone>650-627-1001</phone>
  <defaultshipping>F</defaultshipping>
  <addrtext>Netsuite2955 Campus DriveSan Mateo CA 94403United States</addrtext>
  <state>CA</state>
  <addressee>Netsuite</addressee>
  <isresidential>F</isresidential>
  <label>Work</label>
  <city>San Mateo</city>
  <country>US</country>
  <displaystate>California</displaystate>
  <dropdownstate>CA</dropdownstate>
  <addr1>2955 Campus Drive</addr1>
  <override>F</override>
  <defaultbilling>F</defaultbilling>
</addressbook>

```

## Item Record Format

**i Note:** The format for item pricing varies according to the related features that are enabled in your account. See [Item Pricing Formats](#) for examples.

### JSON

```
{
  "salesdescription": "Cat 5 Patch Cable 10 ft",
  "vendorname": "CABL0002-64",
  "averagecost": 3.50,
  ...
  ... more fields...,

  "pricing":
  [
  ...
  {
    "currency":
    {
      "name": "British pound",
      "internalid": "2"
    },
    "pricelist":
    [
      {
        "pricelevel":
        {
          "name": "Alternate Price 1",
          "internalid": "2"
        },
        "price":
        [
          {
            "price": 9.03,
            "quantitylevel": "1",
            "quantity": 0
          },
          {
            "price": 8.55,
            "quantitylevel": "2",
            "quantity": 10
          }
        ],
        "discount":
        {
          "name": "-5.0%",
          "value": "-5.0%"
        }
      },
      {
        "pricelevel":
        ...
      }
    ]
  }
}
```

```

    {
        "name":"Alternate Price 2",
        "internalid":"3"
    },
    "price":
    [
        {
            "price":8.55,
            "quantitylevel":"1",
            "quantity":0
        },
        {
            "price":8.10,
            "quantitylevel":"2",
            "quantity":10
        }
    ],
    "discount":
    {
        "name": "-10.0%",
        "value": "-10.0%"
    }
},
...
]
}
Repeat for other currencies
],


"productfeed":["FROOGLE","SHOPPING","SHOPZILLA","NEXTAG","YAHOO"],
"weight":"1",
"itemid":"Cable - Cat 5, 10 ft",

... more fields...,

"availabletopartners":false,
"sitecategory":
[
 {"categorydescription":"Cables",
 "category":"12",
 "isdefault":false},
 ],
 "costingmethoddisplay":"Average",
 "offersupport":true
}

```

## XML

```

<salesdescription>Cat 5 Patch Cable 10 ft</salesdescription>
<vendorname>CABL0002-64</vendorname>

```

```
... more ...

<excludefromsitemap>F</excludefromsitemap>
<isdonationitem>F</isdonationitem>
<recordtype>inventoryitem</recordtype>
<createddate>10/12/2006 8:37 pm</createddate>
<cost>3.50</cost>
<price4>
  <pricename>Base Price</pricename>
</price4>
<price4>
  <pricename>Alternate Price 3</pricename>
</price4>
<price4>
  <discountdisplay>-10.0%</discountdisplay>
  <pricename>Corporate Discount Price</pricename>
</price4>
<price4>
  <discountdisplay>-15.0%</discountdisplay>
  <pricename>Employee Price</pricename>
</price4>
<price4>
  <pricename>Online Price</pricename>
</price4>
<price3>
  <pricename>Base Price</pricename>
</price3>
<price3>
  <pricename>Alternate Price 3</pricename>
</price3>
<price3>
  <discountdisplay>-10.0%</discountdisplay>
  <pricename>Corporate Discount Price</pricename>
</price3>
<price3>
  <discountdisplay>-15.0%</discountdisplay>
  <pricename>Employee Price</pricename>
</price3>
<price3>
  <pricename>Online Price</pricename>
</price3>
<price2>
  <price[1]>5.00</price[1]>
  <pricename>Base Price</pricename>
  <price[2]>4.00</price[2]>
</price2>
<price2>
  <pricename>Alternate Price 3</pricename>
</price2>
<price2>
  <discountdisplay>-10.0%</discountdisplay>
  <price[1]>4.50</price[1]>
  <pricename>Corporate Discount Price</pricename>
```

```

<price[2]>3.60</price[2]>
</price2>
<price2>
  <discountdisplay>-15.0%</discountdisplay>
  <price[1]>4.25</price[1]>
  <pricelevelname>Employee Price</pricelevelname>
  <price[2]>3.40</price[2]>
</price2>
<price2>
  <pricelevelname>Online Price</pricelevelname>
</price2>
<price1>
  <price[1]>10.95</price[1]>
  <pricelevelname>Base Price</pricelevelname>
  <price[2]>10.00</price[2]>
</price1>
<price1>
  <pricelevelname>Alternate Price 3</pricelevelname>
</price1>
<price1>
  <discountdisplay>-10.0%</discountdisplay>
  <price[1]>9.86</price[1]>
  <pricelevelname>Corporate Discount Price</pricelevelname>
  <price[2]>9.00</price[2]>
</price1>
<price1>
  <discountdisplay>-15.0%</discountdisplay>
  <price[1]>9.31</price[1]>
  <pricelevelname>Employee Price</pricelevelname>
  <price[2]>8.50</price[2]>
</price1>
<price1>
  <price[1]>10.95</price[1]>
  <pricelevelname>Online Price</pricelevelname>
  <price[2]>10.00</price[2]>
</price1>
<productfeed>FROOGLE</productfeed>
<productfeed>SHOPPING</productfeed>
<productfeed>SHOPZILLA</productfeed>
<productfeed>NEXTAG</productfeed>
<productfeed>YAHOO</productfeed>
<weight>1</weight>
<itemid>Cable - Cat 5, 10 ft</itemid>

... more fields...

<sitecategory>
  <category>12</category>
  <categorydescription>Cables</categorydescription>

  <isdefault>F</isdefault>
</sitecategory>
<offersupport>T</offersupport>
```

## Item Pricing Formats

The format for item pricing varies according to which of the following features are enabled in your account: Multiple Prices, Quantity Pricing, and Multiple Currencies. The following examples show the JSON format for item pricing when these features are enabled.

- Single Price (no additional pricing features enabled)
- Multiple Prices Only Enabled
- Quantity Pricing Only Enabled
- Multiple Prices, Multiple Currencies Enabled
- Multiple Prices, Quantity Pricing, Multiple Currencies Enabled

### Single Price (no additional pricing features enabled)

```
"pricing":  
[  
  {  
    "pricelist":  
      [  
        {  
          "price":  
            [  
              {"price":100.00,"quantitylevel":"1","quantity":0}  
            ]  
        }  
      ],  
      "currency":{"name":"USA","internalid":"1"}  
  }  
]
```

### Multiple Prices Only Enabled

```
"pricing":  
[  
  {  
    "pricelist":  
      [  
        {  
          "pricelevel": {"name": "Base Price", "internalid": "1"},  
          "price":  
            [  
              {"price":100.00,"quantitylevel":"1","quantity":0}  
            ]  
        },  
        {  
          "pricelevel": {"name": "Alternate Price 1", "internalid": "2"},  
          "price":  
            [  
              {"price":99.00,"quantitylevel":"1","quantity":0}  
            ]  
        },  
        {  
          "pricelevel": {"name": "Alternate Price 2", "internalid": "3"},  
          "price":  
            [  
              {"price":98.00,"quantitylevel":"1","quantity":0}  
            ]  
        }  
      ]  
  }  
]
```

```
{
  "pricelist": {
    "name": "Alternate Price 2",
    "internalid": "3"
  },
  "price": [
    {
      "price": 98.00,
      "quantitylevel": "1",
      "quantity": 0
    }
  ],
  "name": "Alternate Price 3",
  "internalid": "4"
},
{
  "pricelist": {
    "name": "Online Price",
    "internalid": "5"
  },
  "price": [
    {
      "price": 96.00,
      "quantitylevel": "1",
      "quantity": 0
    }
  ],
  "name": "USA",
  "internalid": "1"
}
]
```

## Quantity Pricing Only Enabled

```
"pricing": [
  {
    "pricelist": [
      {
        "pricelist": {
          "name": "Base Price",
          "internalid": "1"
        },
        "price": [
          {
            "price": 100.00,
            "quantitylevel": "1",
            "quantity": 0
          },
          {
            "price": 95.00,
            "quantitylevel": "2",
            "quantity": 100
          },
          {
            "price": 90.00,
            "quantitylevel": "3",
            "quantity": 150
          },
          {
            "price": 85.00,
            "quantitylevel": "4",
            "quantity": 200
          },
          {
            "price": 80.00,
            "quantitylevel": "5",
            "quantity": 250
          }
        ]
      },
      {
        "pricelist": {
          "name": "Alternate Price 1",
          "internalid": "2"
        },
        "price": [
          {
            "price": 99.00,
            "quantitylevel": "1",
            "quantity": 0
          },
          {
            "price": 94.00,
            "quantitylevel": "2",
            "quantity": 100
          },
          {
            "price": 89.00,
            "quantitylevel": "3",
            "quantity": 150
          },
          {
            "price": 84.00,
            "quantitylevel": "4",
            "quantity": 200
          },
          {
            "price": 79.00,
            "quantitylevel": "5",
            "quantity": 250
          }
        ]
      }
    ]
  }
]
```

```

        ]
    },
{
    "pricelevel": {"name": "Alternate Price 2", "internalid": "3"},
    "price":
    [
        {"price": 98.00, "quantitylevel": "1", "quantity": 0},
        {"price": 93.00, "quantitylevel": "2", "quantity": 100},
        {"price": 88.00, "quantitylevel": "3", "quantity": 150},
        {"price": 83.00, "quantitylevel": "4", "quantity": 200},
        {"price": 78.00, "quantitylevel": "5", "quantity": 250}
    ]
},
{
    "pricelevel": {"name": "Alternate Price 3", "internalid": "4"},
    "price":
    [
        {"price": 97.00, "quantitylevel": "1", "quantity": 0},
        {"price": 92.00, "quantitylevel": "2", "quantity": 100},
        {"price": 87.00, "quantitylevel": "3", "quantity": 150},
        {"price": 82.00, "quantitylevel": "4", "quantity": 200},
        {"price": 77.00, "quantitylevel": "5", "quantity": 250}
    ]
},
{
    "pricelevel": {"name": "Online Price", "internalid": "5"},
    "price":
    [
        {"price": 96.00, "quantitylevel": "1", "quantity": 0},
        {"price": 91.00, "quantitylevel": "2", "quantity": 100},
        {"price": 86.00, "quantitylevel": "3", "quantity": 150},
        {"price": 81.00, "quantitylevel": "4", "quantity": 200},
        {"price": 76.00, "quantitylevel": "5", "quantity": 250}
    ]
},
],
"currency": {"name": "USA", "internalid": "1"}
}
]

```

## Multiple Prices, Multiple Currencies Enabled

```

"pricing":
[
    {
        "pricelist":
        [
            {
                "pricelevel": {"name": "Base Price", "internalid": "1"},
                "price": [{"price": 110.00, "quantitylevel": "1", "quantity": 0}]
            },
            {
                "pricelevel": {"name": "Alternate Price 1", "internalid": "2"},


```



```

        "price": [{"price": 105.00, "quantitylevel": "1", "quantity": 0}]
    },
    {
        "pricelevel": {"name": "Alternate Price 2", "internalid": "3"},
        "price": [{"price": 100.00, "quantitylevel": "1", "quantity": 0}]
    },
    {
        "pricelevel": {"name": "Alternate Price 3", "internalid": "4"},
        "price": [{"price": 95.00, "quantitylevel": "1", "quantity": 0}]
    },
    {
        "pricelevel": {"name": "Online Price", "internalid": "5"},
        "price": [{"price": 90.00, "quantitylevel": "1", "quantity": 0}]
    }
],
"currency": {"name": "British pound", "internalid": "2"}
},
{
    "pricelist":
    [
        {"pricelevel": {"name": "Base Price", "internalid": "1"}, "price": [{"price": 100.00, "quantitylevel": "1", "quantity": 0}]},
        {"pricelevel": {"name": "Alternate Price 1", "internalid": "2"}, "price": [{"price": 99.00, "quantitylevel": "1", "quantity": 0}]},
        {"pricelevel": {"name": "Alternate Price 2", "internalid": "3"}, "price": [{"price": 98.00, "quantitylevel": "1", "quantity": 0}]},
        {"pricelevel": {"name": "Alternate Price 3", "internalid": "4"}, "price": [{"price": 97.00, "quantitylevel": "1", "quantity": 0}]},
        {"pricelevel": {"name": "Online Price", "internalid": "5"}, "price": [{"price": 96.00, "quantitylevel": "1", "quantity": 0}]}
    ],
    "currency": {"name": "USA", "internalid": "1"}
}
}

```

## Multiple Prices, Quantity Pricing, Multiple Currencies Enabled

```

"pricing":
[
    {
        "pricelist":
        [
            {
                "pricelevel": {"name": "Base Price", "internalid": "1"},
                "price":
                [
                    {"price": 110.00, "quantitylevel": "1", "quantity": 0},
                    {"price": 105.00, "quantitylevel": "2", "quantity": 100},
                    {"price": 100.00, "quantitylevel": "3", "quantity": 150},
                    {"price": 95.00, "quantitylevel": "4", "quantity": 200},
                    {"price": 90.00, "quantitylevel": "5", "quantity": 250}
                ]
            },
            {

```

```

    "pricelevel": {"name": "Alternate Price 1", "internalid": "2"},  

    "price":  

    [  

        {"price": 105.00, "quantitylevel": "1", "quantity": 0},  

        {"price": 100.00, "quantitylevel": "2", "quantity": 100},  

        {"price": 95.00, "quantitylevel": "3", "quantity": 150},  

        {"price": 90.00, "quantitylevel": "4", "quantity": 200},  

        {"price": 85.00, "quantitylevel": "5", "quantity": 250}  

    ]  

},  

{  

    "pricelevel": {"name": "Alternate Price 2", "internalid": "3"},  

    "price": [{"price": 100.00, "quantitylevel": "1", "quantity": 0}, {"price": 95.00, "quantitylevel": "2", "quantity": 100}, {"price": 90.00, "quantitylevel": "3", "quantity": 150}, {"price": 85.00, "quantitylevel": "4", "quantity": 200}, {"price": 80.00, "quantitylevel": "5", "quantity": 250}]  

},  

{  

    "pricelevel": {"name": "Alternate Price 3", "internalid": "4"},  

    "price": [{"price": 95.00, "quantitylevel": "1", "quantity": 0}, {"price": 90.00, "quantitylevel": "2", "quantity": 100}, {"price": 85.00, "quantitylevel": "3", "quantity": 150}, {"price": 80.00, "quantitylevel": "4", "quantity": 200}, {"price": 75.00, "quantitylevel": "5", "quantity": 250}]  

},  

{  

    "pricelevel": {"name": "Online Price", "internalid": "5"},  

    "price": [{"price": 90.00, "quantitylevel": "1", "quantity": 0}, {"price": 85.00, "quantitylevel": "2", "quantity": 100}, {"price": 80.00, "quantitylevel": "3", "quantity": 150}, {"price": 75.00, "quantitylevel": "4", "quantity": 200}, {"price": 70.00, "quantitylevel": "5", "quantity": 250}]  

},  

],  

"currency": {"name": "British pound", "internalid": "2"}  

},  

{  

    "pricelist":  

    [  

        {  

            "pricelevel": {"name": "Base Price", "internalid": "1"},  

            "price": [{"price": 100.00, "quantitylevel": "1", "quantity": 0}, {"price": 95.00, "quantitylevel": "2", "quantity": 100}, {"price": 90.00, "quantitylevel": "3", "quantity": 150}, {"price": 85.00, "quantitylevel": "4", "quantity": 200}, {"price": 80.00, "quantitylevel": "5", "quantity": 250}]  

        },  

        {  

            "pricelevel": {"name": "Alternate Price 1", "internalid": "2"},  

            "price": [{"price": 99.00, "quantitylevel": "1", "quantity": 0}, {"price": 94.00, "quantitylevel": "2", "quantity": 100}, {"price": 89.00, "quantitylevel": "3", "quantity": 150}, {"price": 84.00, "quantitylevel": "4", "quantity": 200}, {"price": 79.0, "quantitylevel": "5", "quantity": 250}]  

        },  

        {  

            "pricelevel": {"name": "Alternate Price 2", "internalid": "3"},  

            "price": [{"price": 98.0, "quantitylevel": "1", "quantity": 0}, {"price": 93.00, "quantitylevel": "2", "quantity": 100}, {"price": 88.00, "quantitylevel": "3", "quantity": 150}, {"price": 83.00, "quantitylevel": "4", "quantity": 200}, {"price": 78.00, "quantitylevel": "5", "quantity": 250}]  

        },  

        {  

            "pricelevel": {"name": "Alternate Price 3", "internalid": "4"},  


```

```

        "price": [{"price":97.00,"quantitylevel":"1","quantity":0},{"price":92.00,"quant
itylevel":"2","quantity":100}, {"price":87.00,"quantitylevel":"3","quantity":150}, {"price":82.00
,"quantitylevel":"4","quantity":200}, {"price":77.00,"quantitylevel":"5","quantity":250}]
    },
    {
        "pricellevel": {"name": "Online Price", "internalid": "5"},
        "price": [{"price":96.00,"quantitylevel":"1","quantity":0}, {"price":91.00,"quant
itylevel":"2","quantity":100}, {"price":86.00,"quantitylevel":"3","quantity":150}, {"price":81.00
,"quantitylevel":"4","quantity":200}, {"price":76.00,"quantitylevel":"5","quantity":250}]
    }
],
"currency": {"name": "USA", "internalid": "1"}
}
]

```

## Sales Order Record Format

### JSON

```
{
    "total":64.04,
    "altshippingcost":5.67,
    "taxtotal":4.45,
    "tranid": "120",
    "orderstatus": "E",
    "shipcomplete": false,
    "discounttotal": 0.00,
    "entity": "76",
    "billaddress": "Doug Fabre\r\nChess\r\nChess Art Gallery\r\n150 N Ocean Dr\r\nMonterey CA 939
40",
    "salesrep": "-5",
    "ccapproved": false,
    "linkedtrackingnumbers": ["1Z6753YA0394527573", "1Z6753YA0394249981"],
    "shipmethod": "92",
    "exchangerate": 1.00
    "lastmodifieddate": "1/9/2011 11:34 pm",
    "taxrate": "8.25%",
    "id": "769",
    "shipaddresslist": "55",
    "istaxable": true,
    "tobefaxed": false,
    "altsalestotal": 0.00,
    "getauth": false,
    "tobeprinted": false,
    "shippingcost": 5.67,
    "recordtype": "salesorder",
    "trandate": "10/14/2006",
    "fax": "831-555-5230",
    "customform": "88",
    "links":
    [
        {"trandate": "10/14/2006", "tranid": "8", "type": "Item Fulfillment", "linktype": "Receipt/Fulfi
    ]
}
```



```

    "llment"}
],
"taxitem": "-112",
"custbody1": "831-555-5229",
"custbody2": "Do not leave the item outside the house",
"shipdate": "10/14/2006",
"createddate": "10/14/2006 2:58 pm",
"subtotal": 53.92,
"currencyname": "USA",
"revenuestatus": "A",
"saleseffectivedate": "10/14/2006",
"email": "chessart@christyscatering.com",
"item":
[
{
  "isclosed": false, "fromjob": false, "amount": 8.96, "rate": 8.96, "price": "2",
  "istaxable": "T", "description": "10 ft Serial Cable DB25M DB25F",
  "custcol6": 429, "custcol7": 2.5,
  "item": "46", "quantity": 1, "iseestimate": false, "commitinventory": "1",
  "options":
  [
    {
      "CUSTCOL3": 792, "CUSTCOL1": 4
    }
  ],
  {
    "isclosed": false, "fromjob": false, "amount": 44.96, "rate": 44.96, "price": "2",
    "istaxable": true,
    "item": "80", "quantity": 1, "iseestimate": false, "commitinventory": "1"
  }
],
"excludecommission": false,
"shipaddress": "Chess\\nChess Art Gallery\\n150 N Ocean Dr\\nMonterey CA 93940", "tobeemail
ed": false
}

```

## XML

```

<altshippingcost>5.67</altshippingcost>
<total>64.04</total>
<taxtotal>4.45</taxtotal>
<orderstatus>E</orderstatus>
<tranid>120</tranid>
<shipcomplete>F</shipcomplete>
<discounttotal>0.00</discounttotal>
<entity>76</entity>
<billaddress>Doug FabreChessChess Art Gallery150 N Ocean DrMonterey CA 93940</billaddress>
<salesrep>-5</salesrep>
<linkedtrackingnumbers>1Z6753YA0394527573</linkedtrackingnumbers>
<linkedtrackingnumbers>1Z6753YA0394249981</linkedtrackingnumbers>
<ccapproved>F</ccapproved>
<shipmethod>92</shipmethod>
<exchangerate>1.00</exchangerate>
<lastmodifieddate>1/9/2011 11:34 pm</lastmodifieddate>

```

```

<taxrate>8.25</taxrate>
<id>769</id>
<shipaddresslist>55</shipaddresslist>
<istaxable>T</istaxable>
<tobefaxed>F</tobefaxed>
<altsalestotal>0.00</altsalestotal>
<getauth>F</getauth>
<tobeprinted>F</tobeprinted>
<shippingcost>5.67</shippingcost>
<recordtype>salesorder</recordtype>
<trandate>10/14/2006</trandate>
<fax>831-555-5230</fax>
<customform>88</customform>
<custbody1>831-555-5229</custbody1>
<custbody2>Do not leave the item outside the house</custbody2>
<shipdate>10/14/2006</shipdate>
<taxitem>-112</taxitem>
<links>
  <trandate>10/14/2006</trandate>
  <tranid>8</tranid>
  <type>Item Fulfillment</type>
  <linktype>Receipt/Fulfillment</linktype>
</links>
<createddate>10/14/2006 2:58 pm</createddate>
<subtotal>53.92</subtotal>
<currencyname>USA</currencyname>
<revenuestatus>A</revenuestatus>
<saleseffectivedate>10/14/2006</saleseffectivedate>
<email>chessart@christyscatering.com</email>
<excludecommission>F</excludecommission>
<item>
  <amount>8.96</amount>
  <fromjob>F</fromjob>
  <isclosed>F</isclosed>
  <price>2</price>
  <rate>8.96</rate>
  <description>10 ft Serial Cable DB25M DB25F</description>
  <istaxable>T</istaxable>
  <item>46</item>
  <quantity>1</quantity>
  <commitininventory>1</commitininventory>
  <custcol6>429</custcol6>
  <custcol7>2.5</custcol7>
  <isestimate>F</isestimate>
  <options>
    <CUSTCOL3>792</CUSTCOL3>
    <CUSTCOL1>4</CUSTCOL1>
  </options>
</item>
<item>
  <amount>44.96</amount>
  <fromjob>F</fromjob>
  <isclosed>F</isclosed>
  <price>2</price>

```

```

<rate>44.96</rate>
<istaxable>T</istaxable>
<item>80</item>
<quantity>1</quantity>
<commitinventory>1</commitinventory>
<isestimate>F</isestimate>
</item>
<shipaddress>ChessChess Art Gallery150 N Ocean DrMonterey CA 93940</shipaddress>
<tobeemailed>F</tobeemailed>

```

## RESTlet Status Codes and Error Message Formats

For details about RESTlet errors, see:

- Success Code
- Error Codes
- Notes about RESTlet Errors
- Error Message Formatting
- Error for Incorrect URL

For information about system errors, see *SuiteScript Errors*.

## Success Code

RESTlets support the following HTTP success code:

- **200 OK:** The RESTlet request was executed successfully.  
The actual response depends on the request method used. For a GET request, the response contains an entity corresponding to the requested resource. For a POST request the response contains an entity describing or containing the result of the action

## Error Codes

RESTlets support the following HTTP error codes:

- **302 Moved Temporarily:** The request was sent to a different data center than the data center in which your company's account resides. When you receive a 302 response, you must recalculate the signature on the request to the correct data center, because the signature is also computed from URL.
- **400 BAD\_REQUEST:** The RESTlet request failed with a user error.
- **401 UNAUTHORIZED:** There is not a valid NetSuite login session for the RESTlet calls.
- **403 FORBIDDEN:** RESTlet request sent to invalid domain, meaning a domain other than https://rest.netsuite.com.
- **404 NOT\_FOUND:** A RESTlet script is not defined in the RESTlet request.
- **405 METHOD\_NOT\_ALLOWED:** The RESTlet request method is not valid.
- **415 UNSUPPORTED\_MEDIA\_TYPE:** An unsupported content type was specified. (Only JSON and text are allowed.)

- **500 INTERNAL\_SERVER\_ERROR (unexpected errors):** Occurs for non-user errors that cannot be recovered by resubmitting the same request.  
If this type of error occurs, contact Customer Support to file a case.
- **503 SERVICE\_UNAVAILABLE:** The NetSuite database is offline or a database connection is not available.

For additional information about HTTP status codes, see [http://www.w3schools.com/tags/ref\\_httpmessages.asp](http://www.w3schools.com/tags/ref_httpmessages.asp)

## Notes about RESTlet Errors

- Any errors encountered at run time that are unhandled return a 400 error. If the user code catches the error, a 200 error is returned.
- An unexpected error is returned with an error ID, for example:

```
Code = UNEXPECTED_ERROR
Msg = An unexpected error occurred. Error ID: fevsjhv41tji2uy3le73
```

- An INVALID\_REQUEST error is returned due to malformed syntax in the OAuth header. For example, when the signature method, version, or timestamp parameters are rejected.
- An INVALID\_LOGIN\_ATTEMPT error is returned when the nonce, consumer key, token, or signature in the OAuth header is invalid.
- The DELETE method is not expected to return anything. In this case, the message is returned:

```
Return was ignored in DELETE operation.
```

- If users specify a content type other than JSON or TEXT, a 415 error is returned with the following message:

```
Invalid content type. You can only use application/json, application/xml or text/plain with RESTlets.
```

- If users provide data in a format different from specified type, the following error is returned with one of the following messages:

```
Error code = INVALID_RETURN_DATA_FORMAT
Error message = Invalid data format. You should return TEXT.
Error message = Invalid data format. You should return a JavaScript object.
```

## Error Message Formatting

The following examples show RESTlet error message formatting for JSON and text content types.

### JSON

```
{
  "error": {
    }
```



```

    "code":"SSS_INVALID_SCRIPTLET_ID",
    "message":"That Suitelet is invalid, disabled, or no longer exists."
}
}

```

## XML

```

<error>
  <code>SSS_INVALID_SCRIPTLET_ID</code>
  <message>That Suitelet is invalid, disabled, or no longer exists.</message>
</error>

```

## Text

```

<error code: SSS_INVALID_SCRIPTLET_ID
error message: That Suitelet is invalid, disabled, or no longer exists.

```

## Error for Incorrect URL

If you receive the following error, make sure that the URL is correct and that it points to the correct RESTlet script ID.

**SSS\_INVALID\_SCRIPTLET\_ID:** That Suitelet is invalid, disabled, or no longer exists.

## Tracking and Managing RESTlet Activity

If you use token-based authentication, you have the ability to track calls that were made by external applications to RESTlets hosted in your NetSuite account. You can track RESTlet activity by using integration records.

When using token-based authentication (TBA), you create an integration record to represent each external application that calls RESTlets. You use the integration record to generate the consumer key and secret needed by the application to authenticate. You can also use the record to do the following:

- Block requests that reference the record's consumer key. You can block requests by setting the record's State field to Blocked.
- Enable or disable token-based authentication for an external application. Note that an integration record can also be used to track web services requests. An additional authentication option on the record, User Credentials, applies only to web services requests. Checking or clearing the User Credentials box has no effect on whether the application can call RESTlets. The User Credentials option affects only web services calls.

Integration records are located at Setup > Integration > Manage Integrations. The record can be accessed only by administrative users.

For full details on using the integration record to set up token-based authentication, see the help topic [Token-based Authentication](#).

For more information on using integration records in conjunction with RESTlets, see the following topics:

- [Ownership of Integration Records](#)
- [Using the RESTlets Execution Log](#)
- [Finding System Notes for an Integration Record](#)
- [More Information](#)

## Ownership of Integration Records

When you create an integration record, it is automatically available to you in your NetSuite account. Your NetSuite account is considered to be the owner of the integration record, and the record is fully editable by administrators in your account.

You can also install records in your account that were created elsewhere. For example, an integration record could be bundled, distributed, and installed outside the account where it was created. If you install a bundle that includes an integration record, the record is considered to be an installed record. It is owned by a different NetSuite account. On such records, you can make changes to only two fields: the Note field and the State field. All other fields, including the authentication and Description fields, can be changed only by an authorized user in the account that owns the record. When the owner makes changes to these fields, the new settings are pushed automatically to your account. These changes are not reflected in the system notes that appear in your account.

## Using the RESTlets Execution Log

Each integration record includes a subtab labeled RESTlets Execution Log. This log lists RESTlet calls that are uniquely identified with that integration record. That is, the log includes those requests that use token-based authentication and reference the integration record's consumer key.



**Note:** Calls made using the NLAuth method of authentication are not logged on any integration record.

For each logged request, the RESTlets Execution Log includes details such as the following:

- The date and time that the call was made.
- The duration of the request.
- The email address of the user who made the request.
- The action taken.
- The corresponding script ID and deployment ID.

## Tracking Changes to Integration Records

If you want to review changes that were made to an integration record, you can refer to the system notes for that record. System notes are used to track events such as the creation of the record, the initial values of its fields, and subsequent updates. For example, if a user changed the State field from Blocked to Enabled, a system note would provide a record of that change.

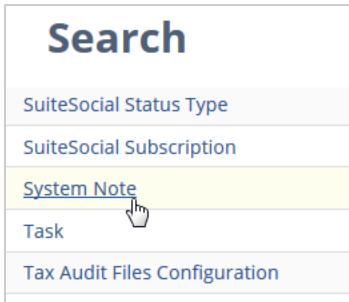
For each event, the system records details such as the ID of the user who made the change and the timestamp of the change. If a user assigns a value to a field that already had a value, the system note also shows the field's former setting.



Be aware that in general, system notes are created only for those fields that you are permitted to change. For additional details, see the help topic [Special Cases Related to System Notes Logging](#). Be aware that this section is part of the SuiteTalk (Web Services) Platform Guide. Some of the detail in this guide pertains only or primarily to web services.

You can locate system notes for integration records in either of the following ways:

- By using the System Note search type, at Reports > New Search.



- By clicking on the System Notes subtab of any integration record.

System Notes						
FIELD	VIEW	- All -	Default			
<b>Customize View</b>						
DATE	SET BY	CONTEXT	TYPE	FIELD	OLD VALUE	NEW VALUE
12/28/2015 1:48 pm	Kate Johnson	UI	Change	State	Enabled	Blocked
12/28/2015 1:48 pm	Kate Johnson	UI	Change	Last State Change By	Jack Smith	Kate Johnson
12/28/2015 12:38 pm	Susan Jones	UI	Set	Note	See Susan for issues related to this integration.	
12/28/2015 12:38 pm	Susan Jones	UI	Set	Description	This integration is used to manage sales orders and customers.	
12/28/2015 12:33 pm	Jack Smith	UI	Set	Token-Based Authentication	F	

## More Information

For more information about managing integration records, see the help topic [Managing Integrations](#), which is part of the SuiteTalk (Web Services) Platform Guide. Be aware that some of the detail in this guide pertains only or primarily to web services:

- [Adding an Integration Record](#)
- [Regenerating a Consumer Key and Secret](#)
- [Distributing by Bundling](#)
- [Using Integration Records in Sandbox Accounts](#)
- [Removing Integration Records](#)
- [Special Cases Related to System Notes Logging](#)

# Scheduled Scripts

- Overview of Scheduled Script Topics
- What Are Scheduled Scripts?
- When Will My Scheduled Script Execute?
- Deploying a Script to the Scheduling Queue
- Creating Multiple Deployments for a Scheduled Script
- Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue
- Understanding Scheduled Script Deployment Statuses
- Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus
- Executing a Scheduled Script in Certain Contexts
- Setting Recovery Points in Scheduled Scripts
- Understanding Memory Usage in Scheduled Scripts
- Monitoring a Scheduled Script's Runtime Status
- Monitoring a Scheduled Script's Governance Limits
- Scheduled Script Samples
- Scheduled Script Best Practices

## Overview of Scheduled Script Topics

The following topics are covered in this section. If you are not familiar with NetSuite scheduled scripts, you should read these topics. They do not need to be read in order. However, if you are learning about NetSuite scheduled scripts, it is strongly recommended that you read the general overview topics first.

### General overview of scheduled scripts

- What Are Scheduled Scripts?
- When Will My Scheduled Script Execute?

### Deploying scripts to NetSuite's scheduling queue

- Deploying a Script to the Scheduling Queue
- Creating Multiple Deployments for a Scheduled Script
- Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue
- Understanding Scheduled Script Deployment Statuses
- Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus

### Scheduled script optimization, recovery points, and monitoring

- Executing a Scheduled Script in Certain Contexts



- Setting Recovery Points in Scheduled Scripts
- Understanding Memory Usage in Scheduled Scripts
- Monitoring a Scheduled Script's Runtime Status
- Monitoring a Scheduled Script's Governance Limits

## Scheduled script samples and best practices

- Scheduled Script Samples
- Scheduled Script Best Practices



**Important:** All companies that run NetSuite are provided a **single** queue for running their scheduled scripts. You can upgrade your number of scheduled script queues from one to five by purchasing NetSuite's SuiteCloud Plus license. See [Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus](#) for more information.

## What Are Scheduled Scripts?

Scheduled scripts run on the NetSuite server. Compared to all other NetSuite script types, scheduled scripts are given a higher limit of usage governance (10,000 units, as opposed to 1,000 units for most other script types). Therefore, scheduled scripts are ideal for long running tasks and batch jobs.

You can deploy scheduled scripts to the NetSuite scheduling queue on an ad-hoc (on-demand) basis. You can also deploy scheduled scripts to the scheduling queue at a future time, or recurring future times.

NetSuite allows you to deploy scheduled scripts to the scheduling queue using the scheduled script Script Deployment page in the UI. You can also call `nlapiScheduleScript` to deploy a script into the scheduling queue.

To understand when your script will execute after it has been deployed to the scheduling queue, see [When Will My Scheduled Script Execute?](#)



**Note:** Scheduled Scripts will not run on a Test Drive account unless the account is a SuiteCloud Developer Network (SDN) Demo Test Drive account.



**Important:** Be aware that scheduled scripts that do mass modifications of data may prompt the system to generate automatic email notifications. For example, if the data being modified is on an Activity record that is set to automatically send an email each time the record is updated, an email will be sent when the scheduled script runs and data is updated.

## When Will My Scheduled Script Execute?

Whether you place a scheduled script into the NetSuite scheduling queue manually (through the Script Deployment page in the UI) or programmatically using `nlapiScheduleScript`, once a script goes into the queue, it is executed serially (first in, first out) on a per-company basis. There is a single queue used by **all** scheduled scripts in your company's NetSuite account. As soon as one script completes, the next script in the queue can be executed. Although multiple scheduled scripts can exist in the queue, only a single script can be executed at any given time.



**Important:** Even when there are no scripts currently in the scheduling queue, and you then place a script into the queue, there may be a short system delay before your script is executed.

When you use the Script Deployment page to schedule the deployment of a script, the times you set on the Schedule subtab are the times the script is being deployed to the scheduling queue. **The times you set on the Schedule subtab are not necessarily the times the script will execute.**

#### Example

The figure below highlights a scheduled deployment of the Load File Sample script. Starting on February 13, 2013 the script will be deployed to the NetSuite scheduling queue at 10:00 pm, and then every 15 minutes after that until midnight of the same day. Script deployment does not mean the script will actually execute precisely at 10:00 pm, 10:15 pm, 10:30 pm, and so on. It means the script will be deployed into the queue at these times.

Again, *when* your script executes depends on how many scripts are ahead of it in the queue. It also depends on how long-running each script before it might be.

To learn more about how to deploy a script into the scheduling queue, see [Deploying a Script to the Scheduling Queue](#).

**Script Deployment**

Save | Cancel | Reset | Change ID | Actions ▾

SCRIPT  
Load File Sample

TITLE \*  
Load File Sample

ID  
customdeploy1

DEPLOYED

STATUS \*  
Scheduled

SEE INSTANCES  
Status Page

LOG LEVEL  
Audit

EXECUTE AS ROLE  
Administrator

**Schedule • Execution Log • History •**

SINGLE EVENT  
 DAILY EVENT    Repeat every  day(s)  
 WEEKLY EVENT    Repeat every weekday  
 MONTHLY EVENT  
 YEARLY EVENT

START DATE \*  
10.1.2011

START TIME  
6:00 pm

REPEAT  
Every 15 minutes

END BY

NO END DATE

Save | Cancel | Reset | Change ID | Actions ▾



**Important:** All companies using NetSuite are provided a **single** queue for running their scheduled scripts. You can upgrade your company's number of scheduled script queues from one to five by purchasing NetSuite's SuiteCloud Plus license. See [Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus](#) for more information.

## Deploying a Script to the Scheduling Queue

You can deploy a script to the scheduling queue right away. Alternatively, you can schedule the deployment so that the script is placed into the queue at a scheduled time in the future, or recurring times in the future.



**Important:** Even if you do an ad-hoc deployment of a script and place it into the queue right away, this does not mean the script will execute right away. Once a script goes into the scheduling queue, there may be a short system delay before the script is actually executed, if no scripts are before it in the queue. If there are scripts already in the queue waiting to be executed, the script deployed into the queue must wait to be executed until all other scripts have completed.

See these topics for details on deploying a script to the NetSuite scheduling queue:

- Initiating an Ad-hoc Deployment of a Script into the Scheduling Queue
- Initiating a Scheduled Deployment of a Script into the Scheduling Queue



**Note:** You can also create **multiple** ad-hoc or future deployments of the same script. There are specific use cases for why you may want to create multiple deployments for the same scheduled script. See [Creating Multiple Deployments for a Scheduled Script](#) to learn more.

## Initiating an Ad-hoc Deployment of a Script into the Scheduling Queue

Scheduled scripts can be deployed into the NetSuite scheduling queue on an ad-hoc (on-demand) basis. To do this, you will use the **Save and Execute** command on the Script Deployment page. The Status field on the Script Deployment page must be set to either **Not Scheduled** or **Testing**.

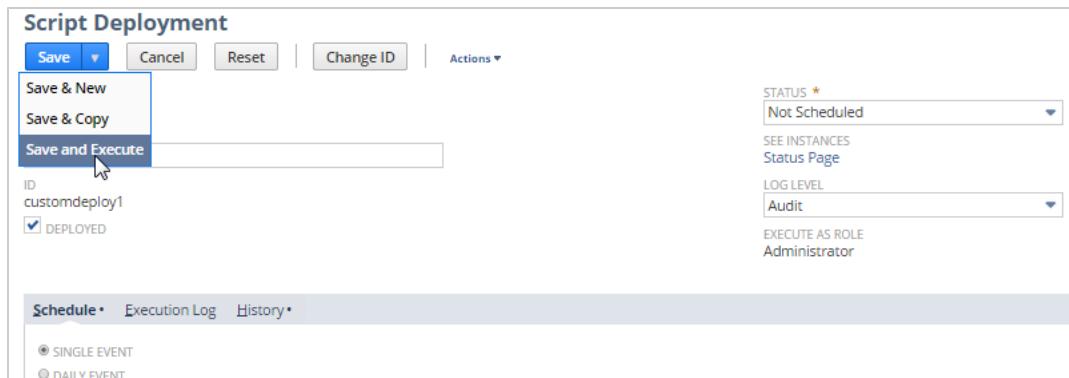


**Note:** To understand why you may want to set the deployment status to either Not Scheduled or Testing for an ad-hoc deployment, see [Understanding the Difference Between Not Scheduled and Testing Deployment Statuses](#).

Deployments set to **Not Scheduled** can also be deployed into the queue on-demand through a call to nlapiScheduleScript. For details, see [Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue](#).

### To initiate an ad-hoc deployment of a script into the scheduling queue:

1. After creating your SuiteScript JavaScript file, create a new Script record for your script. Go to Customization > Scripting > Scripts > New > Scheduled.
2. On the Script page, provide a **Name** for the Script record.
3. On the **Scripts** tab, select your script file from the **Script File** drop-down and specify the script's executing function.
4. Next, click **Save**.
5. Click the **Deploy Script** button on the page that appears.
6. When the Script Deployment page opens (see figure), click the **Deployed** check box if it is not already checked.
7. Select **Not Scheduled** (or **Testing**) from the **Status** field.
8. Click the **Single Event** radio button.
9. Next, click **Save and Execute**.



**Important:** Even if you do an ad-hoc deployment of a script and place it into the queue right away, this does not mean the script will execute right away. Once a script goes into the scheduling queue, there may be a short system delay before the script is actually executed, if no scripts are before it in the queue. If there are scripts already in the queue waiting to be executed, the script deployed into the queue must wait to be executed until all other scripts have completed.

## Understanding the Difference Between Not Scheduled and Testing Deployment Statuses

To deploy a scheduled script on-demand, set the deployment status to either **Not Scheduled** or **Testing** for the following reasons:

Deployment Status	Use Case
Testing	<p>Set to Testing for the following reasons:</p> <ul style="list-style-type: none"> <li>■ You want to test the script by running it immediately. The script will run only in the script owner's account. After setting the deployment status to Testing, click <b>Save and Execute</b> on the Script Deployment page to run the script.</li> <li>■ You want to load the script into the SuiteScript Debugger. Only scheduled scripts with the deployment status set to Testing can be loaded into the Debugger.</li> </ul>
Not Scheduled	<p>Set to Not Scheduled after all of the scheduling options have been set (time, date, frequency); however, the script is not yet ready to be executed.</p> <p><b>Important:</b> If you set your scheduling options, set the deployment status to Not Scheduled, and click <b>Save</b>, the script will not run at the times you have specified.</p> <p>Scripts with a deployment status set to Not Scheduled will only run if you click <b>Save and Execute</b> (or the scripts are placed into the NetSuite scheduling queue through a call to <code>napiScheduleScript</code>.)</p>

### Notes:

- If you set the deployment status to **Not Scheduled**, and then select either Daily Event, Weekly Event (or any event type other than **Single Event**), and click **Save and Execute**, the script will still only run once.

- After a **Not Scheduled** script completes its ad-hoc execution, NetSuite automatically re-sets the deployment status back to **Not Scheduled**.

## Initiating a Scheduled Deployment of a Script into the Scheduling Queue

Scheduled scripts with a deployment status set to **Scheduled** can be deployed to the scheduling queue once at a pre-defined time in the future, or their deployments can be scheduled on a regular daily, weekly, monthly, or yearly basis.

Deployment times can be scheduled with a frequency of every 15 minutes, for example 2:00 pm, 2:15 pm, 2:30 pm, and so on.



**Important:** If the deployment status of a script is set to **Scheduled**, you cannot call `nlapIScheduleScript` to put the script into the scheduling queue.

A scheduled script's deployment status should be set to **Scheduled** for the following reasons:

- The script was set to **Testing**, but is now ready for production.
- The script does not need to be executed immediately.
- The script must run at recurring times.

### To initiate a scheduled deployment of a script:

- After creating your SuiteScript JavaScript file, create a new Script record for your script. Go to Customization > Scripting > Scripts > New > Scheduled.
- On the Script page, provide a **Name** for the Script record.
- On the **Scripts** tab, select your script file from the **Script File** drop-down, and specify the script's function.
- Next, click **Save**.
- Click the **Deploy Script** button on the page that appears.
- When the Script Deployment page opens, click the **Deployed** check box if it is not already checked, and select **Scheduled** from the **Status** drop-down.
- On the **Schedule** tab, set all deployment options.
- Click **Save**.



**Note:** You can also create **multiple** ad-hoc or future deployments of the same script. There are specific use cases for why you may want to create multiple deployments for the same scheduled script. See [Creating Multiple Deployments for a Scheduled Script](#) to learn more.

## Creating Multiple Deployments for a Scheduled Script

On either the Script page or the Script Deployment page you can create multiple deployments for the same script. See these sections for more information:

- [Use Cases for Creating Multiple Deployments](#)
- [Steps for Creating Multiple Deployments](#)
- [Multiple Deployments and `nlapIScheduleScript`](#)



**Note:** You can set unique deployment options for scheduled scripts on either the scheduled script **Script** page or the **Script Deployment** page.

## Use Cases for Creating Multiple Deployments

The following use cases describe possible scenarios in which script owners may want to create multiple deployments for a single scheduled script.

- Use Case 1 - Same Script Requires Different Deployment Schedules
- Use Case 2 - Same Script Receives Multiple Requests for Execution
- Use Case 3 - Script May Exceed Unit-based Governance Limits

### Use Case 1 - Same Script Requires Different Deployment Schedules

Create multiple deployments when you want the same script to execute according to different deployment schedules.

For example, you might have a scheduled script that you want deployed and executed on the last day of every month. This would be your first deployment. You might also want this script to be deployed and executed every Monday morning around 2:00 am. This would be your second, separate deployment for this script.

### Use Case 2 - Same Script Receives Multiple Requests for Execution

There may be times when you have concurrent (simultaneous) requests to kick off long-running tasks. In situations like this, you want to be able to schedule the next available deployment for each request that comes in. For example, if you think that at peak load you might receive five requests in a given time frame, then you would want at least that many script deployments available to ensure that each request gets its own dedicated script deployment.

Note that if one of the deployments already happens to be in progress or in the queue, the best practice is to take the number of requests you expect over some time period (for example, over a five minute period - which is more than the average script execution time) and then multiply by two to get the number of script deployments you would need to ensure that you can handle the load. In this case you would want to create 10 deployments for the same script.

### Use Case 3 - Script May Exceed Unit-based Governance Limits

Another reason to create multiple deployments is if you have a script you think will exceed governance limits. After creating different deployments, you can specify a different deployment through the `deployId` parameter in [nlapiScheduleScript\(scriptId, deployId, params\)](#).



**Note:** See [SuiteScript Governance](#) for information on scheduled script governance limits.



**Important:** All companies that run NetSuite are provided a **single** queue for running their scheduled scripts. You can upgrade your number of scheduled script queues from one to five by purchasing NetSuite's SuiteCloud Plus license. See [Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus](#) for more information.

## Steps for Creating Multiple Deployments

### To schedule multiple deployments for the same script:

1. On the Script record page > **Deployments** tab, set your deployment options (see first figure).

2. Click **Add** after setting your deployment values.
3. Create another deployment (if necessary).
4. Optionally, create your own unique deployment ID in the **ID** column. If you do not add your own custom deployment ID (see figure), an ID is automatically generated.
5. When finished creating all deployments, click **Save**.

The screenshot shows the 'Script' page with the 'Deployments' tab selected. The page displays the following details:

- TYPE:** Scheduled
- NAME \***: Long Running Task
- ID**: customscript\_long\_running\_task
- DESCRIPTION**: (empty)
- OWNER**: Jay Ramos
- INACTIVE**: (unchecked)

SCRIPTS	PARAMETERS	UNHANDLED ERRORS	EXECUTION LOG	DEPLOYMENTS	HISTORY
Long Running Task 2	customdeploy2	Yes	Not Scheduled	Debug	one time event on 6/17/2014
Long Running Task	customdeploy_long_dep_1	Yes	Not Scheduled	Debug	one time event on 6/17/2014

To edit or access each deployment, go to Customization > Scripting > Script Deployments.

### Creating additional deployments from the same deployment:

You can create additional script deployments from existing deployments. This is useful for quickly adding extra deployments for scheduled scripts.

1. On the Script Deployments page, click **View** on an existing deployment.
2. From the **More Actions** dropdown click **Make Copy**.
3. Check the details for the deployment, correcting where necessary.
4. Click **Save**.



**Note:** Make Copy is only available for scheduled scripts.

## Multiple Deployments and nlapiScheduleScript

You can create multiple on-demand deployment instances and parameterize the call to `nlapiScheduleScript`. Users can queue-up multiple instances of the same **Not Scheduled** script. The `nlapiScheduleScript` API will call the first of the script deployments that is ready and in the queue, and continue to call this same script until each instance of the script has been deployed. This means that users can queue as many instances of the **Not Scheduled** script through User Event scripts as they have deployments.

In this scenario, it is not recommended that users set the `deployId` parameter in `nlapiScheduleScript` since the deployments are generally "ad-hoc" and created real-time.

**Note:** For more information on nlapiScheduleScript, see [Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue](#). Also see the API documentation for nlapiScheduleScript(scriptId, deployId, params).

## Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue

You can programmatically deploy a scheduled script into your company's scheduling queue using the nlapiScheduleScript API. Be sure that the script's deployment status appears as **Not Scheduled** on the Script Deployment page.

Using nlapiScheduleScript you can:

- Place a currently executing scheduled script back into the scheduling queue
- Call another scheduled script from within a scheduled script. When the new script is called, it is then put into the scheduling queue.
- Place a scheduled script into the queue from another script such as a user event script or a suitelet.

**Note:** Scheduled scripts called by nlapiScheduleScript(scriptId, deployId, params) may show a NULL status if the script that was called has not yet been deployed or does not exist in NetSuite.

The ability to call nlapiScheduleScript from within a scheduled script allows developers to **automatically** place their currently executing script back into the scheduling queue. Otherwise, scheduled scripts must be re-queued manually through the Script Deployment page. See [Example 2 in Scheduled Script Samples](#) for code that shows how to programmatically re-queue a scheduled script.

Developers should call nlapiScheduleScript in a scheduled script if they think the script is coming close to exceeding the 10,000 unit limit allotted to scheduled scripts. The call to nlapiScheduleScript will place the script back in the queue, and the script can then run to completion without exceeding any governance limits.

Note that if nlapiScheduleScript is used in a scheduled script to call **another** scheduled script, instruction count limits are applied to **each** script separately, since (technically) you are running two different scheduled scripts. In other words, both "scheduled script A" and "scheduled script B," which was called by "scheduled script A" can **each** contain 10,000 units.

**Note:** See [SuiteScript Governance](#) for information on unit-based governance limits.

## Understanding Scheduled Script Deployment Statuses

The following describes each of the deployment statuses of a scheduled script deployment. These deployment statuses appear in the Status field of the **Script Deployment** page.

- **Not Scheduled:** Means that the script is not currently scheduled to go into the queue. By clicking Save and Execute on the Script Deployment page, scheduled scripts with a Not Scheduled

deployment status are placed ad-hoc (on-demand) into the scheduling queue. Scripts with a Not Scheduled deployment status can also be placed into the queue programmatically through a call to nlapiScheduleScript.

- **Scheduled** : Means that the script will be deployed into the queue at the time(s) specified on the Schedule subtab of the Script Deployment page. If the deployment is set to happen on a recurring basis, the script's deployment status will remain as **Scheduled**, even after the script completes its execution. The script will then be re-deployed to the scheduling queue at its specified time(s).
- **Testing** : Means that when the scheduled script is executed, it will run only in the script owner's account. Note that when the deployment status is set to Testing, the only way to place the script into the scheduling queue is by clicking Save and Execute on the Script Deployment page. You cannot schedule testing times and then click Save.
  - Also note that only scheduled scripts with a deployment status set to Testing can be loaded into the SuiteScript Debugger.

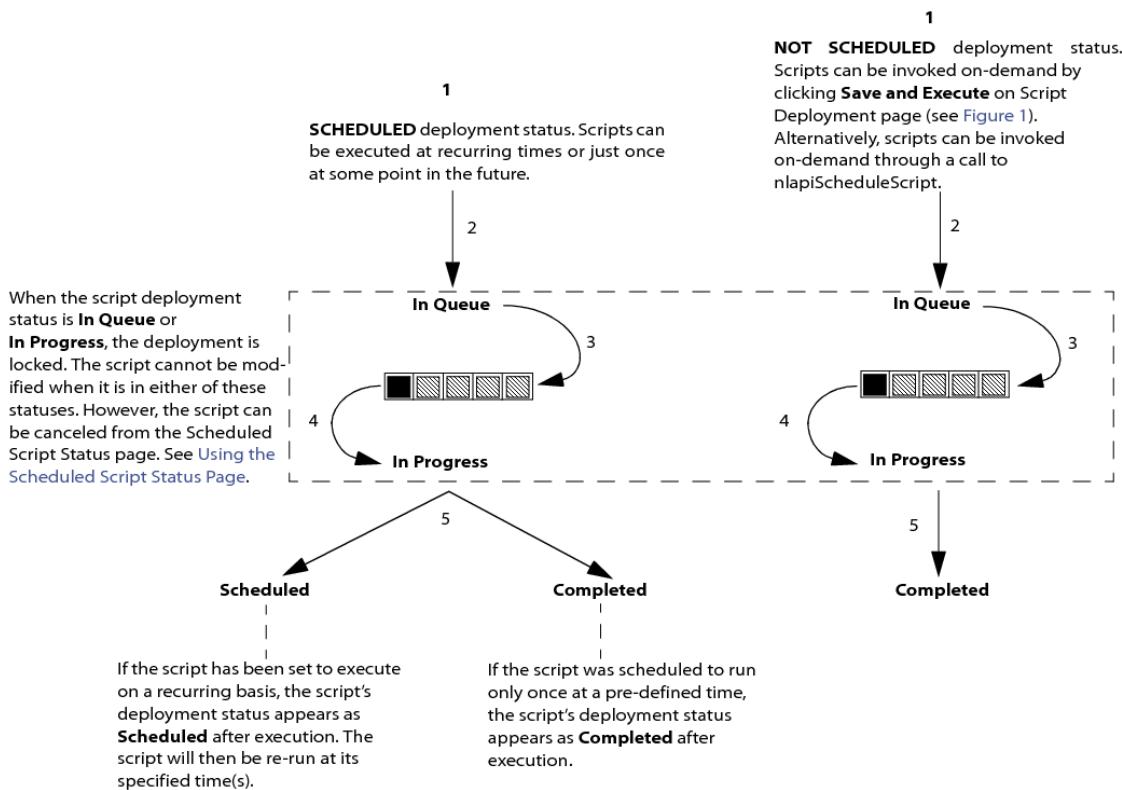
This table summarizes what you can and cannot do with a scheduled script depending the script's deployment status on the Script Deployment page.

The **second column** states whether the script can be placed into the NetSuite scheduling queue through the UI.

The **third column** states whether the script can be queued using nlapiScheduleScript.

Deployment Status	UI	nlapiScheduleScript
Not Scheduled	YES. You must click the Save and Execute button on the Script Deployment page.	YES. See <a href="#">Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue</a> for details.
Scheduled	N/A. No action is required by the user. With a deployment status set to Scheduled, the script will run again at the next scheduled time.	NO
Testing	YES. On the Script Deployment page users must click Save and Execute if they want to run the script for testing purposes. You cannot schedule testing times and then click Save. Only Save and Execute will run a script that has a Testing status. Also note that only scheduled scripts with a deployment status set to Testing can be loaded into the SuiteScript Debugger. Finally, scheduled scripts set to Testing will run in the script owner's account only.	NO

A script's **deployment** status set to **Scheduled** or **Not Scheduled** determines its path of execution in NetSuite.



## Executing a Scheduled Script in Certain Contexts

When creating a scheduled script, you can associate a **type** argument that is passed by the system to the script's executing function. The **type** argument provides a context for when the scheduled script is invoked.



**Important:** The **type** argument is an auto-generated argument passed by the system. You cannot set this as a parameter for a specific deployment like other function arguments.

Valid values for the **type** argument are:

- **scheduled** - normal execution according to the deployment options specified in the UI
- **ondemand** - the script is executed via a call to `nlapiScheduleScript`
- **userinterface** - the script is executed via the UI (the Save & Execute button has been clicked)
- **aborted** - re-executed automatically following an aborted execution (system went down during execution)
- **skipped** - executed automatically following downtime during which the script should have been executed

### Example

```
function processOrdersCreatedToday( type )
```

```
{
//only execute when run from the scheduler, based on the deployment options set in the UI
if ( type != 'scheduled' && type != 'skipped' ) return ;

.... //process rest of script

}
```

## Setting Recovery Points in Scheduled Scripts

Occasionally while running a scheduled script a failure may occur. This could be due to a major NetSuite upgrade, or an unexpected failure of the execution environment. Therefore, NetSuite gives developers the ability to create recovery points in scheduled scripts. These recovery points allow the state of the script at a certain point to be saved. In the event of an unexpected system failure, the script can be restarted from the last successful recovery point.

To set a script recovery point, use [nlapiSetRecoveryPoint\(\)](#). When the system restarts, the script will resume where it left off.

Developers can also use [nlapiYieldScript\(\)](#). In addition to setting a recovery point, this API places the script back into the scheduled script queue. Once the script moves to the front of the queue for processing, it begins its execution from the specified recovery point.

Possible use cases for [nlapiSetRecoveryPoint\(\)](#) and [nlapiYieldScript\(\)](#) include:

- If the script is unexpectedly aborted, it can be restarted from the last successful recovery point.
- Pause (yield) the execution of a script at a specified point.
- Governance usage limits have been reached.
- Yield a script because an external resource is temporarily unavailable.

See the API documentation on [nlapiSetRecoveryPoint\(\)](#) and [nlapiYieldScript\(\)](#) for more details and example usage.

## Understanding Memory Usage in Scheduled Scripts

The memory limit for a scheduled script is 50 Megabytes. Therefore, if you have a long-running scheduled script, and you are concerned the script will exceed the 50 MB memory limit, NetSuite recommends using [nlapiSetRecoveryPoint\(\)](#) or [nlapiYieldScript\(\)](#) APIs to track memory size. This is accomplished by examining the returned size property in the status object returned by [nlapiSetRecoveryPoint\(\)](#) and [nlapiYieldScript\(\)](#). Note, however, calling [nlapiSetRecoveryPoint\(\)](#) costs 100 governance units. Therefore, you will want to use the API only at key points in your script. The alternative approach is to use [nlapiYieldScript\(\)](#). If the call is successful, the script will yield and then the size property can be examined after the script resumes.



**Important:** Scripts that are resumed after an `nlapiYieldScript()` or `nlapiSetRecoveryPoint()` call will have their governance units reset. However, this does not reset the memory footprint of the script, which will continue to change until the script terminates. Therefore it is possible for the script to stay under the governance limit but exceed the memory size limit, at which point an `SS_EXCESSIVE_MEMORY_FOOTPRINT` error is thrown during the call to `nlapiYieldScript()` or `nlapiSetRecoveryPoint()`.

## Reducing Script Memory Footprint

The table below shows an estimation of how various parts of a script can consume memory.

Empty Function	111 bytes
Each Instruction within a function	32 bytes
Each local variable reference	32 bytes
Standard mode record (customer)	40 kilobytes (depends on record type)
Dynamic mode record (customer)	460 kilobytes (depends on record type)
Number Instance	32 bytes
String Instance	32 bytes + 4 bytes per character
Empty Object	32 bytes

There are several useful techniques to reduce the memory overhead of scripts. Certain returned objects, for example `nlobjRecord` and `nlobjSearchResult` can be quite large. In order to reduce the memory consumed by these objects, convert them to native JavaScript objects and then operate on those objects instead.

## Monitoring a Scheduled Script's Runtime Status

The Scheduled Script Status page shows the current and past runtime statuses of all scheduled scripts that have been executed in your account. There are three different ways you can access the Scheduled Script Status page. How you access the page determines the view that the page opens with.

### Use the Setup Menu

Access the Scheduled Script Status page using the Setup menu option. Go to Customization > Scripting > Script Deployments > Status.

This view will show all scheduled scripts in your accounts and each deployment of the script. You can use filtering options at the bottom of the page to see the runtime status of specific scripts and deployments.

Scheduled Script Status							
<input type="button" value="Refresh"/> <input type="button" value="FILTERS"/>							
SCRIPT	DEPLOYMENT ID	DATE CREATED ▾	STATUS	START	END	% COMPLETE	CANCEL
Tax Bundle Maintenance	customdeploy_tax_bundle_maintenance	10/29/2014 12:02.58 am	Complete	12:02.58 am	12:02.59 am	100.0%	
Tax Bundle Maintenance	customdeploy_tax_bundle_maintenance	10/29/2014 12:02.27 am	Complete	12:02.57 am	12:02.58 am	100.0%	
ITR Generate Field Cache	customdeploy_generate_field_cache	10/29/2014 12:01.35 am	Complete	12:02.28 am	12:02.57 am	100.0%	
Tax Bundle Maintenance	customdeploy_tax_bundle_maintenance	10/29/2014 12:01.35 am	Complete	12:02.11 am	12:02.27 am	100.0%	
Tax Bundle Maintenance	customdeploy_tax_bundle_maintenance	10/28/2014 7:33.14 pm	Complete	7:33.15 pm	7:33.15 pm	100.0%	



**Important:** Script execution details are purged after 30 days. Also note that the statuses listed on the Scheduled Script Status page are not related to the statuses that appear on a Script Deployment page. The statuses on Script Deployment pages (Not Scheduled, Scheduled, Testing) indicate the type of deployment. The statuses on the Scheduled Script Status page indicate where the scheduled script is in terms of its execution.

## Use the Status Page or Status Links

You can also access the Scheduled Script Status page by clicking the Status Page link (associated with the See Instances field). This link appears on the Script Deployment page. See figure below.

Script Deployment	
<input type="button" value="Edit"/>	<input type="button" value="Back"/>
<input type="button" value="Actions ▾"/>	
SCRIPT	STATUS
Tax Bundle Maintenance	Not Scheduled
TITLE	<b>SEE INSTANCES</b>
Tax Bundle Maintenance	<b>Status Page</b>
ID	LOG LEVEL
customdeploy_tax_bundle_maintenance	Error
<input checked="" type="checkbox"/> DEPLOYED	EXECUTE AS ROLE
	Administrator

In both cases, when the Scheduled Script Status page opens, it shows the runtime statuses of all deployed instances of a particular script.

If you want to see the runtime statuses of other deployments of a particular script, use the **Deployment ID** filter to choose another deployment of this script. You can also choose ALL to see the runtime statuses of every scheduled script deployment of this script.

If you want to see the runtime statuses of other scripts, use the **Script** filter to choose another script. Then use the **Deployment ID** filter to specify which deployments of the script you want to see the runtime status for.

Scheduled Script Status						
<input type="button" value="Refresh"/> <b>FILTERS</b>						
DATE	FROM	TO	SCRIPT	DEPLOYMENT ID		
All			Payment Batch Processing	CUSTOMDEPLOY_266...H_PROCESSING_SS		
- All -	CUSTOMDEPLOY_2663_BATCH_PROCESSING_SS		.10.2014 — 20.10.2014		<input type="button" value="&lt;"/>	<input type="button" value="&gt;"/>
DEPLOYMENT ID	DATE			CUSTOMDEPLOY_2663_BATCH_PROCESSING_SS2	END	% COMPLETE
customdeploy_2663_batch_processing_ss	28.10.2014 11:02:08 pm	Complete	11:02:33 pm	11:03:01 pm	100.0%	
customdeploy_2663_batch_processing_ss	28.10.2014 10:01:34 pm	Complete	10:02:25 pm	10:02:56 pm	100.0%	
customdeploy_2663_batch_processing_ss	28.10.2014 9:01:53 pm	Complete	9:02:32 pm	9:03:45 pm	100.0%	
customdeploy_2663_batch_processing_ss	28.10.2014 8:01:44 pm	Complete	8:02:22 pm	8:03:36 pm	100.0%	
customdeploy_2663_batch_processing_ss	28.10.2014 7:02:08 pm	Complete	7:02:49 pm	7:03:33 pm	100.0%	
customdeploy_2663_batch_processing_ss	28.10.2014 6:02:16 pm	Complete	6:02:38 pm	6:02:52 pm	100.0%	
customdeploy_2663_batch_processing_ss	27.10.2014 11:02:14 pm	Complete	11:02:35 pm	11:03:46 pm	100.0%	

## Monitoring a Scheduled Script's Governance Limits

The following APIs are helpful when working with and monitoring scheduled scripts:

- `napiLogExecution(type, title, details)`
- `nlobjContext` methods:
  - `setPercentComplete(pct)`
  - `getPercentComplete()`
  - `getRemainingUsage()`
  - `getscriptId()`
  - `getDeploymentId()`

## Deploying Scheduled Scripts to Multiple Queues Through SuiteCloud Plus

All companies that run NetSuite are provided a single queue for running their scheduled scripts. Companies can upgrade their number of scheduled script queues from one to **five** with the purchase of a SuiteCloud Plus license. The purchase of two SuiteCloud Plus licenses provides 10 queues and the purchase of three licenses provides 15 queues. SuiteCloud Plus allows larger accounts to divide their scheduled script work into categories such as script type, script length, department, and so on.

For a summary of SuiteCloud Plus capabilities, see the help topic [Using SuiteCloud Plus](#).

When you upgrade your account to include multiple script queues, the scripts in each queue will execute serially. For example, if there are two scripts in Queue 1, the first script must complete before the second script begins. Across all queues, the scripts will execute concurrently. For example, if you have one script in each of five queues, all scripts will run at the same time.

After purchasing one or more SuiteCloud Plus licenses, you will notice that a **Queue** dropdown field is added to the Script Deployment record for scheduled scripts (see figure). The script author or

administrator can use this field to set the target queue of a script. Note that to process the same script concurrently, scriptors can create multiple deployments for the same script, and then set each deployment to a different queue.

The screenshot shows the 'Script Deployment' page. On the left, there's a list of deployment details:

- SCRIPT:** Sched Script A
- TITLE:** Sched Script A 3
- ID:** customdeploy3
- DEPLOYED:**
- STATUS:** Not Scheduled

On the right, there are several options:

- SEE INSTANCES:** Status Page
- LOG LEVEL:** Debug
- QUEUE:** **1** (This field is highlighted with a red box.)
- EXECUTE AS ROLE:** Administrator

Additionally, a **Queue** column will appear on the Script Deployment Status page to indicate which queue a script has been deployed to. Users can set the **Queue** filter on the bottom of this page to sort scripts based on queue number.

Should you choose to purchase SuiteCloud Plus, be aware that all of your existing scripts will, by default, initially be put into Queue 1. You will need to go the Script Deployment pages of each schedule script to reassign the script to another queue. Also note that if you do not assign a queue number to a new script, the script will automatically be assigned to Queue 1.

To learn more about SuiteCloud Plus, or to purchase it contact your NetSuite account manager.

## Determining Which Queue Your Scheduled Script Is In

After you have purchased one or more SuiteCloud Plus licenses, you can use a **queue** argument to obtain the queue number assigned to a scheduled script. The **queue** argument is the second argument passed to the launch or executing function of a scheduled script. (The first argument is the **type** argument, which you can use regardless of whether you have a SuiteCloud Plus license. For details on the **type** argument, [Executing a Scheduled Script in Certain Contexts](#).)

The **queue** argument provides the id of the queue you selected in the Queue field of the scheduled script's Script Deployment page.



**Important:** The **queue** argument is an auto-generated argument passed by the system. You cannot set this as a parameter for a specific deployment like other function arguments.

If you have purchased a single SuiteCloud Plus license, valid values for the **queue** argument are: 1, 2, 3, 4, 5. If you have purchased two licenses, 6,7,8,9,10 are also valid. If you have purchased three licenses, 11, 12, 13, 14, 15 are also valid.

### Example

```
function processOrdersCreatedToday( type, queue )
{
    //only execute when run from the scheduler and the script is in queue 5,
    // based on the deployment options set in the UI
    if ( type != 'scheduled' && type != 'skipped' && queue == 5 ) return ;

    .... //process rest of script
```

```
}
```

## Scheduled Script Samples

The following scheduled script code samples are provided in this section:

- Example 1 - Fulfill and Bill Sales Orders on a Daily Basis
- Example 2 - Reschedule a Script Depending on Units Remaining
- Example 3 - Create a Drip Marketing Campaign
- Example 4 - Automatically Send 'Thank You' Emails to Valued Customers
- Example 5 - Passing Script Parameters in a Scheduled Script

### Example 1 - Fulfill and Bill Sales Orders on a Daily Basis

This script fulfills and bills all sales orders created today. This is a batch operation that would normally take a long time to execute, making it an ideal candidate for a scheduled script.

```
function processOrdersCreatedToday( type )
{
    //only execute when run from the scheduler
    if ( type != 'scheduled' && type != 'skipped' ) return;

    var filters = new Array();
    filters[0] = new nlobjSearchFilter( 'mainline', null, 'is', 'T' );
    filters[1] = new nlobjSearchFilter( 'trandate', null, 'equalTo', 'today' );

    var searchresults = nlapiSearchRecord( 'salesorder', null, filters, null, new nlobjSearchColumn('terms') );
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
    {
        var id = searchresults[i].getId();
        var fulfillRecord = nlapiTransformRecord('salesorder', id, 'itemfulfillment');
        nlapiSubmitRecord( fulfillRecord );

        var billType = searchresults[i].getValue('paymentmethod') == null ? 'invoice' : 'cashesale';
        var billRecord = nlapiTransformRecord('salesorder', id, billType);
        nlapiSubmitRecord( billRecord );
    }
}
```

### Example 2 - Reschedule a Script Depending on Units Remaining

Use `nlapiScheduleScript`, `nlobjContext.getscriptId()`, and `nlobjContext.getDeploymentId()` to reschedule the currently executing scheduled script if there are more sales orders to update when the unit usage limit is reached.

```
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 3 - Create a Drip Marketing Campaign

This example illustrates a daily scheduled script for processing a drip marketing campaign with two touch points and one branch point. The basic workflow involves:

- Schedule Campaign for new leads (clone and schedule an existing campaign)
- Schedule follow-up Campaign for leads that are seven days old but whose statuses have not changed
- Schedule follow-up phone calls for sales reps assigned to these leads
- Schedule Campaign for leads that are seven days old whose statuses have since been upgraded

#### Parameters & Setup

- custscript\_newleads campaign list parameter containing base campaign used for emailing new leads
- custscript\_weekold campaign list parameter containing base campaign used for emailing week old unchanged leads
- custscriptConverted campaign list parameter containing base campaign used for emailing week old upgraded leads
- custscript\_leadstatus entitystatus list parameter containing the status used to define what a "new" lead is.

```

function processDripMarketing( type )
{
    if ( type != 'scheduled' ) return; /* script should only execute during scheduled calls. */

    /* process new leads */
    scheduleCampaign( custscript_newleads );
    /* process one-week old unchanged leads */
    scheduleCampaign( custscript_weekold );
    /* process follow-up for one-week old unchanged leads */
    scheduleFollowUpPhoneCall();
    /* process follow-up email for one-week old converted leads. */
    scheduleCampaign( custscriptConverted );
}

function scheduleCampaign( base_campaign )
{
    var today = nlapiDateToString( new Date() );
    var campaign = nlapiCopyRecord('campaign', base_campaign);
}

```

```

campaign.setFieldValue('startdate', today);
campaign.setFieldValue('title',campaign.getFieldValue('title') + (' + today + '));

campaign.setLineItemValue('campaignemail','status',1,'EXECUTE');
campaign.setLineItemValue('campaignemail','datescheduled',1,today);
nlapiSubmitRecord( campaign );
}

function scheduleFollowUpPhoneCall()
{
    var filters = new Array();
    filters[0] = new nlobjSearchFilter('datecreated',null,'on','daysago7');
    filters[1] = new nlobjSearchFilter('status',null,'equalto', custscript_leadstatus);

    var columns = new Array();
    columns[0] = new nlobjSearchColumn('salesrep');
    columns[1] = new nlobjSearchColumn('phone');
    columns[2] = new nlobjSearchColumn('entityid');

    var today = nlapiDateToString( new Date() );
    var leads = nlapiSearchRecord('customer',null,filters,columns);
    for ( var i = 0; leads != null && i < leads.length; i++ )
    {
        var leadId = leads[i].getId();
        var salesrep = leads[i].getValue('salesrep');
        var phonenumber = leads[i].getValue('phone');
        var leadName = leads[i].getValue('entityid');
        /* Schedule Phone Call only if the lead is assigned and has a number. */
        if ( salesrep != null && phonenumber != null )
        {
            var call = nlapiCreateRecord('phonecall');
            call.setFieldValue('title','Follow up Call for '+leadName);
            call.setFieldValue('startdate', today );
            call.setFieldValue('assigned', salesrep );
            call.setFieldValue('phone', phonenumber );
            call.setFieldValue('company', leadId );
            call.setFieldValue('status','SCHEDULED');
            nlapiSubmitRecord( call );
        }
    }
}
}

```

#### Example 4 - Automatically Send 'Thank You' Emails to Valued Customers

This sample shows how to create a scheduled script to send thank you notes to valued, repeated customers. A scheduled script may be executed to perform daily searches for sales orders that are placed today and within the last 30 days from the same customer. After retrieving the results, the scheduled script then sends these customers an email on behalf of the sales rep to thank them for their repeated business.

Notice there are a number of `nlobjContext.getRemainingUsage()` API calls in the sample. This API provides the remaining SuiteScript usage to help scripts monitor how close they are to running into SuiteScript usage governance.

```

/*
 * This scheduled script looks for customers that
 * have placed multiple orders in the last 30 days.
 * It will send a thank you email to these customers
 * on behalf of their sales reps.
 */
function findHotCustomerScheduled(type)
{
    //Invoke only when it is scheduled
    if(type == 'scheduled')
    {
        //Obtaining the context object and logging the remaining usage available
        var context = nlapiGetContext();
        nlapiLogExecution('DEBUG', 'Remaining usage at script beginning', context.getRemainingUsage
());
        //Setting up filters to search for sales orders
        //with trandate of today.
        var todaySOFilters = new Array();
        todaySOFilters[0] = new nlobjSearchFilter('trandate', null, 'on', 'today');

        //Setting up the columns. Note the join entity.salesrep column.
        var todaySOCOLUMNS = new Array();
        todaySOCOLUMNS[0] = new nlobjSearchColumn('tranid', null, null);
        todaySOCOLUMNS[1] = new nlobjSearchColumn('entity', null, null);
        todaySOCOLUMNS[2] = new nlobjSearchColumn('salesrep', 'entity', null);

        //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', conte
xt.getRemainingUsage());
        //Looping through each result found
        for(var i = 0; todaySO != null && i < todaySO.length; i++)
        {
            //obtain a result
            var so = todaySO[i];
            //Setting up the filters for another sales order search
            //that are of the same customer and have trandate within
            //the last 30 days
            var oldSOFilters = new Array();
            var thirtyDaysAgo = nlapiAddDays(new Date(), -30);
            oldSOFilters[0] = new nlobjSearchFilter('trandate', null, 'onorafter', thirtyDaysAgo);
            oldSOFilters[1] = new nlobjSearchFilter('entity', null, 'is', so.getValue('entity'));
            oldSOFilters[2] = new nlobjSearchFilter('tranid', null, 'isnot', so.getValue('tranid'));

            //Search for for the repeated sales in the last 30 days
            var oldSO = nlapiSearchRecord('salesorder', null, oldSOFilters, null);
            nlapiLogExecution('DEBUG', 'Remaining usage after in for loop, i=' + i, context.getRemain
ingUsage());
            //If results are found, send a thank you email
        }
    }
}

```

```

if(oldSO != null)
{
    //Setting up the subject and body of the email
    var subject = 'Thank you!';
    var body = 'Dear ' + so.getText('entity') + ', thank you for your repeated business in
the last 30 days.';

    //Sending the thank you email to the customer on behalf of the sales rep
    //Note the code to obtain the join column entity.salesrep
    nlapiSendEmail(so.getValue('salesrep', 'entity'), so.getValue('entity'), subject, body)
;

    nlapiLogExecution('DEBUG', 'Remaining usage after sending thank you email', context.get
RemainingUsage());
}
}
}
}
}

```

### Example 5 - Passing Script Parameters in a Scheduled Script

The following sample shows how to retrieve passed parameters (by calling the `getSetting(...)` method on the `nlobjContext` object) within a scheduled script. It also shows how to execute a scheduled script by passing the custom ID (`scriptId`) of the Script record and the custom deployment ID (`deployId`) of the Script Deployment record. For details on working with script parameters, see [Creating Script Parameters Overview](#).

```

//retrieve parameters inside a scheduled script
function scheduled_main()
{
//get script parameter values
var context = nlapiGetContext();
var strStartDate = context.getSetting('SCRIPT', 'custscriptstartdate');

var subsidiary = context.getSetting('SCRIPT', 'custscriptsubsidiary');
var startDate = new Date(strStartDate);

//schedule the script execution and define script parameter values
var startDate = new Date();
var params = {
    custscriptstartdate: startDate.toUTCString(),
    custscriptsubsidiary: 42
}

//so that the scheduled script API knows which script to run, set the custom ID
//specified on the Script record. Then set the custom ID on the Script Deployment
nlapiScheduleScript('customscript_audit_report', 'customdeploy_audit_report_dp', params);
}

```

**i Note:** Since scheduled scripts also trigger user event scripts, developers may need to revisit the design of their user event scripts to ensure they will be invoked by the correct execution contexts.



# Portlet Scripts

The following topics are covered in this section:

- [What Are Portlet Scripts?](#)
- [Portlet Script Execution](#)
- [Assigning the Portlet Preference to a Script Parameter](#)
- [Running a Portlet Script in NetSuite](#)
- [Displaying Portlet Scripts on the Dashboard](#)
- [Portlet Scripts Samples](#)

## What Are Portlet Scripts?

Portlet scripts can be used to define and publish custom dashboard content. The following portlet types can be created:

- **LIST** - A standard list of user-defined column headers and rows (for example a Search Results portlet). See [List Portlet](#) for an example of a list portlet.
- **FORM** - A basic data entry form with up to one submit button embedded into a portlet (for example a Quickadd portlet). This type of portlet supports APIs to refresh and resize the portlet, as well as the use of record-level client-side script to implement validation. See [Form-level and Record-level Client Scripts](#) for details about this type of script. See [Form Portlet](#) for an example of a form portlet.
- **HTML** - An HTML-based portlet, the most flexible presentation format used to display free-form HTML (images, Flash, custom HTML). See [HTML Portlet](#) for an example of an HTML portlet.
- **LINKS** - This default portlet consists of rows of formatted content (for example an RSS portlet). See [Links Portlet](#) for an example of a links portlet.

Be aware that the portlet type (LIST, FORM, HTML, LINKS) is not actually passed as a value in the portlet script itself, rather it is defined on the portlet Script record page (see figure below). After you create your portlet.js file, you will load your .js file into the file cabinet, create a new script record for your file (Setup > Customization > Scripts > New > Portlet), and then select the portlet type from the Portlet Type drop-down menu.

The screenshot shows the 'Script' creation interface in NetSuite. At the top, there are buttons for 'Save & New' (highlighted in blue), 'Cancel', and 'Reset'. Below these are fields for 'NAME \*' (containing 'My portlet') and 'ID'. On the right, there are fields for 'DESCRIPTION' (empty), 'OWNER' (containing 'Wolfe, K'), and a checked 'INACTIVE' checkbox. A large dropdown menu labeled 'PORTLET TYPE \*' is open, showing options: 'Simple List' (selected and highlighted in blue), 'Inline HTML', 'Links and Indents', 'Simple Form', and 'Deployments' (which is a submenu under 'Simple List'). At the bottom, there are tabs for 'Libraries' and 'Custom Plug-In Types'.

# Portlet Script Execution

Portlet scripts run on the server and are rendered in the NetSuite dashboard. A user-defined portlet function is executed whenever a SuiteScript-generated portlet is opened or refreshed by the user.

When writing portlet scripts, NetSuite automatically passes two arguments to your user-defined function. These arguments are:

- *portlet* - References a `nlobjPortlet` object
- *column* - Column index for this portlet on the dashboard (valid values are: 1 = left column, 2 = middle column, 3 = right column)

For custom portlets on [Customer Dashboards](#), NetSuite can pass the following additional argument:

- *entityid* - References the customer ID for the selected customer.

## Example

```
function mySamplePortlet( portlet, column )
{
    remainder of portlet script...
}
```

Note that *column* is an optional argument. If you choose not to pass a column value in your script, you can write:

```
function mySamplePortlet( portlet )
{
    remainder of portlet script...
}
```

Portlet scripts can only run after users have added the scripts to their dashboards. After the scripts have been added, users must then open their dashboards for a portlet script to execute.

 **Note:** To add portlet scripts to the dashboard, see [Displaying Portlet Scripts on the Dashboard](#).

## Assigning the Portlet Preference to a Script Parameter

Portlet script parameters (custom fields) can be configured to be customizable as portlet settings. This allows users to modify their script parameters for each portlet. For information on setting the *portlet* preference on script parameters, see [Setting Script Parameter Preferences](#). If you are not familiar with the concept of script parameters, see [Creating Script Parameters Overview](#).

## Running a Portlet Script in NetSuite

To run a portlet script in NetSuite, you must:

1. Create a JavaScript file for your portlet script.
2. Load the file into NetSuite.
3. Create a Script record.
4. Define all runtime options on the Script Deployment page.

If you are new to SuiteScript and need information on each of these steps, see [Running Scripts in NetSuite Overview](#).



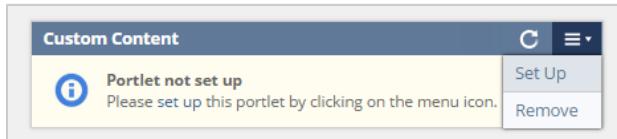
**Important:** Portlets scripts require that you reference the script from a custom portlet. See [Displaying Portlet Scripts on the Dashboard](#) for details.

## Displaying Portlet Scripts on the Dashboard

If you have created a portlet using SuiteScript, use these steps to display the custom portlet on your dashboard. Note that the following steps are to be completed only **after** you have performed all steps in the section [Running a Portlet Script in NetSuite](#).

### To display portlet scripts on the dashboard:

1. Go to your dashboard and click the **Personalize Dashboard** link.
2. Click one of the **Custom Portlet** links under the **Standard Content** folder.  
An empty Custom Content portlet appears on your dashboard.
3. Hover over the Portlet Setup arrow and click **Set Up**.



4. In the Set Up Scripted Content popup, select the desired portlet script from the **Source** drop-down list, and then click **Save**.  
The portlet will populate with data as defined in your portlet script.

## Portlet Scripts Samples

The following sample portlets are provided:

- List Portlet
- Form Portlet
- HTML Portlet
- Links Portlet

The following image shows how the portlet samples appear in NetSuite:

The screenshot shows a SuiteScript interface with a top navigation bar and a 'Home' section. Below are three portlets:

- Flash Portlet:** Displays a logo for 'Christy's Catering'.
- Simple Form Portlet:** Contains fields for TEXT, INTEGER, DATE, SELECT (with options Oranges, Apples, Bananas), and TEXTAREA, with a 'Submit' button.
- Estimates List Detail:** A table listing estimates from EST10001 to EST10013, including columns for Number, Date, Customer, Sales Rep, and Amount.

Item	Description
1	Sample list portlet
2	Sample form portlet
3	Sample HTML portlet

## List Portlet

This script searches for a list of estimates and displays the results in a list format. See [nlobjPortlet](#) for a list of portlet object methods.

### Script:

```
function demoListPortlet(portlet, column)
{
    portlet.setTitle(column != 2 ? "Estimates List" : "Estimates List Detail")
    var col = portlet.addColumn('tranid','text', 'Number', 'LEFT');
    col.setURL(nlapiResolveURL('RECORD','estimate'));
    col.addParamToURL('id','id', true);
    portlet.addColumn('trandate','date', 'Date', 'LEFT');
    portlet.addColumn('entity_display','text', 'Customer', 'LEFT');
    if ( column == 2 )
    {
        portlet.addColumn('salesrep_display','text', 'Sales Rep', 'LEFT');
        portlet.addColumn('amount','currency', 'Amount', 'RIGHT');
    }
    var returncols = new Array();
```

```

returncols[0] = new nllobjSearchColumn('trandate');
returncols[1] = new nllobjSearchColumn('tranid');
returncols[2] = new nllobjSearchColumn('entity');
returncols[3] = new nllobjSearchColumn('salesrep');
returncols[4] = new nllobjSearchColumn('amount');
var results = nlapiSearchRecord('estimate', null, new
    nllobjSearchFilter('mainline',null,'is','T'), returncols);
for ( var i = 0; i < Math.min((column != 2 ? 5 : 15 ),results.length); i++ )
    portlet.addRow( results[i] )
}

```

## Form Portlet

This script builds a very simple form in a portlet that POSTs data to a servlet. This form includes one embedded Submit button.

See [nllobjPortlet](#) for a list of portlet object methods.

### Script:

```

function demoSimpleFormPortlet(portlet, column)
{
    portlet.setTitle('Simple Form Portlet')
    var fld = portlet.addField('text','text','Text');
    fld.setLayoutType('normal','startcol');
    portlet.addField('integer','integer','Integer');
    portlet.addField('date','date','Date');
    var select = portlet.addField('fruit','select','Select');
    select.addSelectOption('a','Oranges');
    select.addSelectOption('b','Apples');
    select.addSelectOption('c','Bananas');
    portlet.addField('textarea','textarea','Textarea');
    portlet.setSubmitButton(nlapiResolveURL('SUITELET','customscript_simpleformbackend', 'custo
mdeploy_simpleform'),'Submit');
}

```

## HTML Portlet

This portlet script generates the HTML required to download and display a FLASH animation in an HTML portlet. See [nllobjPortlet](#) for a list of portlet object methods.

### Script:

```

function demoRichClientPortlet(portlet, column)
{
    portlet.setTitle('Flash Portlet')
    var content = "<table align=center border=0 cellpadding=3 cellspacing=0
width=100%><tr><td>" +
        "<OBJECT CLASSID='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'>" +
        "<PARAM NAME='MOVIE' VALUE='/images/flash/tomato.swf>" +

```



```

        "<embed src='/images/flash/tomato.swf'></embed></OBJECT></td>
        </tr></table>";
    content = '<td><span>' + content + '</span></td>';
    portlet.setHtml( content );
}

```

**i Note:** To use the HTML from a text file uploaded to the file cabinet, substitute the HTML string assigned to var content with `nlapiLoadFile(id).getValue()`.

## Links Portlet

This script makes an external request to slashdot.org to retrieve and display an RSS feed in a LINKS portlet. The APIs used are `nlapiRequestURL(url, postdata, headers, callback, httpMethod)` to fetch the RSS feed, `nlapiStringToXML(text)` to convert the feed into an XML document, and `nlapiSelectNodes(node, xpath) / nlapiSelectValue(node, xpath)` to query the XML document for the RSS data.

See [nlobjPortlet](#) for a list of portlet object methods.

### Script:

```

function demoRssPortlet(portlet)
{
    portlet.setTitle('Custom RSS Feed');
    var feeds = getRssFeed();
    if ( feeds != null && feeds.length > 0 )
    {
        for ( var i=0; i < feeds.length ; i++ )
        {
            portlet.addLine('#'+(i+1)+': '+feeds[i].title, feeds[i].url, 0);
            portlet.addLine(feeds[i].description, null, 1);
        }
    }
}
function getRssFeed()
{
    var url = 'http://rss.slashdot.org/Slashdot/slashdot';
    var response = nlapiRequestURL( url, null, null );
    var responseXML = nlapiStringToXML( response.getBody() );
    var rawfeeds = nlapiSelectNodes( responseXML, "//item" );
    var feeds = new Array();
    for (var i = 0; i < rawfeeds.length && i < 5 ; i++)
    {
        feeds[feeds.length++] = new rssfeed( nlapiSelectValue(rawfeeds[i], "title"),
                                             nlapiSelectValue(rawfeeds[i], "link"),
                                             nlapiSelectValue(rawfeeds[i], "description"));
    }
    return feeds;
}
function rssfeed(title, url, description)
{

```

```
this.title = title;  
this.url = url;  
this.description = description;  
}
```

# Mass Update Scripts

The following topics are covered in this section:

- [What Are Mass Update Scripts?](#)
- [Mass Update Script Execution](#)
- [Running a Mass Update Script in NetSuite](#)
- [Mass Update Scripts Samples](#)

## What Are Mass Update Scripts?

Mass update scripts allow you to programmatically perform custom mass updates to update fields that are not available through general mass updates. You can also use mass update scripts to run complex calculations, as defined in your script, across many records.



**Note:** If you are not familiar with mass update functionality in NetSuite, see the help topic [Making Mass Changes or Updates](#) in the NetSuite Help Center.

When a custom mass update is performed, the **record type** being updated is passed to a system-defined **rec\_type** parameter in the mass update script. Additionally, the **internal ID** of each record in the custom mass update is passed to a system-defined **rec\_id** parameter.

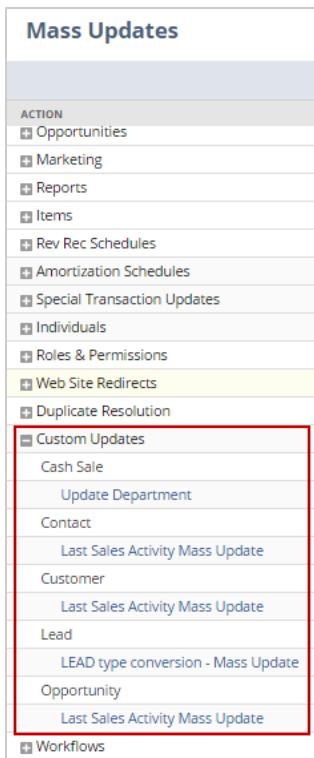
Whether you are using a custom mass update to update fields that are (or are not) available through inline editing, or you are updating fields based on the value of a SuiteScript script parameter, the executing function in your script will include the **rec\_type** and **rec\_id** parameters. For example:

```
function updateMemo(rec_type, rec_id )  
{  
    nlapiSubmitField(rec_type, rec_id, 'memo', 'Premiere Customer', true);  
}
```



**Note:** See [Mass Update Scripts Samples](#) for more details on working with mass update scripts.

Like all other script types, you must create a Script record and a Script Deployment record for mass update scripts. After you define the deployment for a mass update script and specify which record types the script will run against, the record type(s) will appear under the Custom Updates dropdown, accessed by going to Lists > Mass Update > Mass Updates > Custom Updates (see figure).



## Mass Update Script Metering and Governance

The SuiteScript governance limit is 1000 units **per record** /invocation of a mass update script.

Also note that if multiple rows are returned for the same transaction, the custom mass update will still run only once per transaction.

## Creating Script Parameters for Mass Update Scripts

The mass update script type allows you to create script parameters on the Script record page. Script parameters will then appear as fields in the header portion of the custom Mass Update page of the specified record type (see figure). Users executing custom mass updates can set the value(s) of one or more script parameters on the Mass Update page before running the update.

The screenshot shows the 'Mass Update' page in NetSuite. At the top, there are buttons for 'Save', 'Cancel', 'Reset', and 'Preview'. Below these, a 'TITLE OF ACTION \*' field contains 'Update Department on SO Mass Update'. A dropdown menu under 'NEW DEPARTMENT' has 'Service' selected. A tooltip box appears over this dropdown, stating: 'The script parameter called New Department was defined on the mass update script record and can be set by users on the Mass Update page.' To the right, the 'ACTION' is listed as 'updateDepartment' with a checked 'PUBLIC' checkbox. Below the title, tabs for 'Criteria', 'Results', 'Audience', 'Schedule', 'Audit Trail', and 'Action Title Translation' are visible. Under 'Criteria', there is a note: 'Use this tab to specify criteria that narrow down your search.' An unchecked checkbox for 'USE EXPRESSIONS' is present. A 'FILTER \*' table is shown with two rows: 'Created By' (description: 'is - My Team -') and 'Date Created' (description: 'is within last month').

Note that your SuiteScript code must use the `nlobjContext.getSetting(type, name)` method to get the user-defined value of the script parameter. See [Mass Update Scripts Samples](#) for more details on working with script parameters within action scripts.

When you first create your script parameter you can set a parameter value as a **user** or **company** preference. The parameter will default to this value, but users may edit it as they run the mass update.

When you preview a custom mass update, the selected parameters will be shown for reference in the footer of the search results page.



**Note:** If you are not familiar with script parameters in SuiteScript, see [Creating Script Parameters Overview](#) in the NetSuite Help Center.

## Mass Update Script Execution

Mass Update scripts execute on the server. They are not considered to be client scripts that run in the browser.

Mass Update scripts are executed when users click the Perform Update button on the Mass Update Preview Results page.



**Important:** Mass update scripts can only be invoked from the Mass Update page. They cannot be invoked from another script type. For example, you cannot invoke a mass update script by passing the script's `scriptId` and `deployId` to the `nlapiScheduleScript(scriptId, deployId, params)` function.

You have a choice of running mass update scripts as admin or as the logged-in user. As a script owner, you must have the Mass Update permission to test and work with mass update scripts. Users must have the Client SuiteScript and Server SuiteScript features enabled in their accounts for the scripts to run. Also be aware that users who perform the custom mass update need the appropriate permission (Edit or Full) for the record types they are updating.

If a mass update script encounters an error, the script execution will abort. Only the updates that are completed prior to the error will be committed to the database.

Also note that the execution context for a mass update script is `custommassupdate`. This is important if you are trying to determine the execution context of a script using `nlobjContext.getExecutionContext()`.

Finally, be aware that updates made to records during a custom mass update can trigger user event scripts if there are user event scripts associated with the records being updated.

## Running a Mass Update Script in NetSuite

To run a mass update script in NetSuite, you must:

1. Create a JavaScript file for your action script.
2. Load the file into NetSuite.
3. Create a Script record.
4. Define all runtime options on the Script Deployment page.
5. After you define the deployment for a mass update script and specify which record types the script will run against, the record type(s) will appear under the Custom Updates dropdown, accessed by going to Lists > Mass Update > Mass Updates > Custom Updates.

If you are new to SuiteScript and need information on steps 1–4, see [Running Scripts in NetSuite Overview](#).



**Important:** When running mass update scripts in NetSuite, be aware of the following:

- Mass update script deployments and mass updates can both be assigned an audience. It is the script owner's responsibility to ensure the two audiences are in sync. If the two audiences do not match, the mass update script will not run when users click the Perform Update button on the Mass Update page.
- When users run custom mass updates, they must have the appropriate permission (Edit/Full) for the record type(s) they are updating.
- Users must also have SuiteScript enabled in their accounts. (Administrators can go to Setup > Company > Enabled Features > SuiteFlex tab > and click the Server SuiteScript check box and the Client SuiteScript check box.)

## Mass Update Scripts Samples

The following mass update script samples are provided in this section:

- Updating a field that is available through inline edit
- Updating a field that is not available through inline edit
- Updating a field based on a script parameter value

### Updating a field that is available through inline edit

The following sample mass update script shows that the Memo field on every record of a certain type will be updated in a custom mass update. The record type that this script will run against (Sales

Order, for example) is defined on the action Script Deployment page. When the custom mass update is executed by a user, the individual record IDs for **each** record of that type is passed to the mass update script's `rec_id` parameter.

Note that in the UI, the Memo field can be inline editing. In SuiteScript, fields that are inline editable are updated using the `nlapiSubmitField(type, id, fields, values, doSourcing)` function.

```
function updateMemo(rec_type, rec_id)
{
    nlapiSubmitField(rec_type, rec_id, 'memo', 'Premiere Customer', true);
}
```

In the sample above, when the user clicks the Perform Update button on the Mass Update Preview Results page, the Memo field on all specified sales orders will be updated to the text value **Premiere Customer**.



**Important:** Be aware that if the Memo field were not inline editable, you would have to load and submit each record in the custom mass update. The `nlapiSubmitField` function is used only on fields that can be inline edited through the UI. For example mass update scripts that update fields that are not available through inline edit, see [Updating a field that is not available through inline edit](#).

## Updating a field that is not available through inline edit

The second example updates the Probability field on the Opportunity record type. The record type is specified on the Script Deployment page for the action script.

After all custom mass update search criteria are defined on the Mass Update page for the Opportunity record type, this script will run on all Opportunity records that match the criteria. The Probability field will then be updated to the new value.

```
function updProbability(rec_type, rec_id)
{
    var recOpportunity = nlapiLoadRecord(rec_type, rec_id);
    recOpportunity.setFieldValue('probability', 61);
    nlapiSubmitRecord(recOpportunity);
}
```

Note that in this sample, you are required to load and submit the entire record to change the value of the Probability field. You must do this when the field you want to change in the custom mass update cannot be inline edited. In other words, you cannot update the field using `nlapiSubmitField(type, id, fields, values, doSourcing)` without first loading the entire record object.

## Updating a field based on a script parameter value

This sample mass update script updates the Department field on the Sales Order and Estimate record types. The update is based on the value of a script parameter called New Department (`custscript_dept_update`). Note that because the Department field is not accessible through inline editing, the only way to change the value of the Department field is to load and submit each record that the custom mass update is running against.

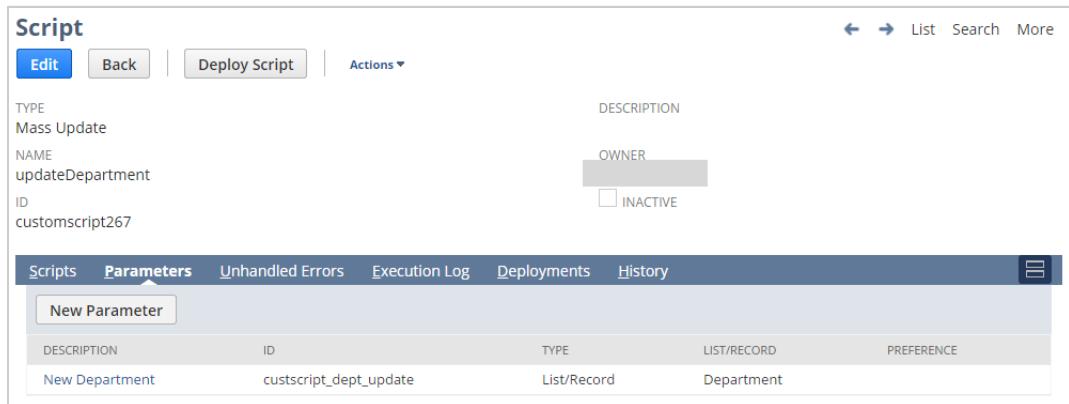
## To perform a custom mass update that references a script parameter:

1. Create an action script (see below for an example). The script below will update the **Department** field on the record types specified in the action script's deployment. The update of the **Department** field will be based on the user-defined value specified in Step 6.

Notice that the script's executing function takes the `rec_type` and `rec_id` parameters. When the custom mass update is performed, the record type defined on the deployment and the internal ID of each record will get passed to the executing function.

```
function updateDepartment(rec_type, rec_id)
{
    var transaction = nlapiLoadRecord(rec_type, rec_id);
    transaction.setFieldValue('department', nlapiGetContext().getSetting('SCRIPT',
        'custscript_dept_update'));
    nlapiSubmitRecord(transaction, false, true);
}
```

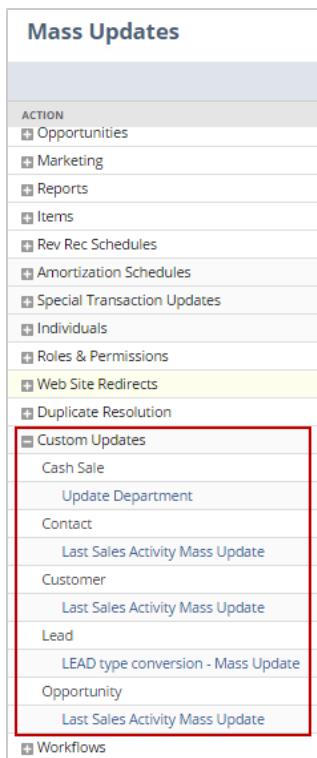
2. Create a mass update Script record and define the new script parameter (see figure).



3. Create a script deployment (see figure below). In this example, the Update Department script will be deployed to Sales Order records.



4. After deploying the mass update script, go to Lists > Mass Update > Mass Updates. All custom mass updates referencing a mass update script will appear under the **Custom Updates** dropdown. The following figure shows that the Update Department script has been deployed to the Estimate and Sales Order record types (see figure).



5. Click the custom mass update you want to execute. In this example, the **Update Department** link under the Sales Order deployment is selected.
6. On the Mass Update page for the record type, specify the value of the script parameter. In this example, the value of the **New Department** script parameter is set to **Services** (see figure).

The screenshot shows the 'Mass Update' page for the 'Sales Order' record type. The 'ACTION' field is set to 'updateDepartment'. The 'NEW DEPARTMENT' dropdown is set to 'Service'. The 'Criteria' tab is selected, showing a table with two rows: 'Created By' and 'Date Created'. The 'Created By' row has a 'DESCRIPTION' column with the value 'is - My Team -' and a 'FORMULA' column with the value 'is within last month'. A red box highlights the 'Criteria' tab and the first two rows of the table.

7. Next, as with any mass update, use tabs on the Mass Update page to:
1. Define which records the custom mass update will apply to (**Criteria** tab).

2. Define which records you want to see when you preview the custom mass update (**Results** tab).
3. Define the Audience that the custom mass update will apply to (**Audience** tab).



**Important:** Be sure that the audience you define on the Mass Update page matches the audience defined on the Script Deployment page.

4. Set the frequency with which you want the custom mass update to run (**Schedule** tab).
8. Click **Preview** to verify which records the custom mass update will apply to.
9. Click **Perform Update** to run the custom mass update.

# Bundle Installation Scripts

The following topics are covered in this section:

- What are Bundle Installation Scripts?
- Setting Up a Bundle Installation Script
- Sample Bundle Installation Script

## What are Bundle Installation Scripts?

Bundle installation scripts are specialized server SuiteScripts that perform processing in target accounts as part of bundle installation, update, or uninstall. This processing can include setup, configuration, and data management tasks that would otherwise have to be completed by account administrators. These scripts enhance solution providers' ability to manage the bundle deployment process.

Every bundle can include a bundle installation script that is automatically run when the bundle is installed, upgraded, or uninstalled. Each bundle installation script can contain triggers to be executed before install, after install, before update, after update, and after uninstall. All triggers should be included in a single script file. This trigger code can ensure that bundles are implemented correctly, and can prevent bundle installation, update, or uninstall if proper setup has not occurred.

Bundle installation scripts have no audience because they are always executed using the administrator role, in the context of bundle installation, update, or uninstall. Bundle installation scripts do not have event types.

A bundle installation script can be associated with multiple bundles. Before a script can be associated with a bundle, it must have a script record and at least one deployment. A bundle creator associates a bundle installation script with a bundle by selecting one of its deployments in the Bundle Builder. The script .js file and script record are automatically included in the bundle when it is added to target accounts. Script file contents can be hidden from target accounts based on an option set for the .js file in the file cabinet record.

## Bundle Installation Script Functions

### Triggered Functions

A bundle installation script's functions are executed automatically during bundle installation, update, or uninstall, based on one or more of the following triggers:

- Before Install - Executed before a bundle is installed for the first time in a target account.
- After Install - Executed after a bundle is installed for the first time in a target account.
- Before Update - Executed before a bundle in a target account is updated.
- After Update - Executed after a bundle in a target account is updated.
- Before Uninstall - Executed before a bundle is uninstalled from a target account.

A bundle installation script file should include a function for at least one of these triggers. If you are using more than one of these, they should all be in the same script file.

The following are example uses for bundle installation script triggered functions:

- Before Install: Check the existing configuration and setup in the target account prior to bundle installation, and halt the installation with an error message if the target account does not meet minimum requirements to run the solution.
- After Install: Automate the setup and configuration of the bundled application after it has been installed in the target account, eliminating manual tasks.
- After Install or After Update: Connect to an external system to fetch some data and complete the setup of the bundled application.
- Before Update: Manage required data changes in the target account prior to executing an upgrade.
- Before Uninstall: Reset configuration settings or remove data associated with the bundle being uninstalled.

## Function Parameters

Two specialized parameters are available to functions in bundle installation scripts, to return the version of bundles, as specified on the Bundle Basics page of the Bundle Builder.

- The **toversion** parameter returns the version of the bundle that will be installed in the target account. This parameter is available to Before Install, After Install, Before Update, and After Update functions.
- The **fromversion** parameter returns the version of the bundle that is currently installed in the target account. This parameter is available to Before Update and After Update functions.

## Getting Bundle Context

### Calls to Functions in Other Script Files

A bundle installation script file can include calls to functions in other script files, as long as these files are added as library script files on the script record. Any .js files for library script files are automatically included in the bundle when it is added to target accounts.

Bundle installation scripts can call scheduled scripts, but only in the After Install and After Update functions. Calls to scheduled scripts are not supported in the Before Install, Before Update, and Before Uninstall functions.

## Bundle Installation Script Governance

Bundle installation scripts are governed by a maximum of 10,000 units per execution.

## Defining Deployments for Bundle Installation Scripts

You can create multiple deployments for each bundle installation script, with different parameters for each, but only one deployment can be associated with each bundle. When you associate a bundle installation script with a bundle, you select a specific script deployment.

Bundle installation scripts need to be executed with administrator privileges, so the Execute as Role field should always be set to Administrator on the script deployment record.

Bundle installation scripts can only be run in target accounts if the Status is set to Released. The Status should be set to Testing if you want to debug the script.

## Bundle Installation Script Error Handling

Any bundle installation script failure terminates bundle installation, update, or uninstall.

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

## Setting Up a Bundle Installation Script

Complete the following tasks to set up a bundle installation script:

- [Create the Bundle Installation Script File](#)
- [Add the Bundle Installation Script File to the File Cabinet](#)
- [Create the Bundle Installation Script Record](#)
- [Define Bundle Installation Script Deployment](#)
- [Associate the Script with a Bundle](#)

A bundle installation script is a specialized server SuiteScript that is executed automatically in target accounts when a bundle is installed, updated, or uninstalled. For details about how to create a bundle, see [Using the Bundle Builder](#) and [Creating a Bundle with the Bundle Builder](#).

### Create the Bundle Installation Script File

You can create a bundle installation script file in the same manner that you create other types of SuiteScript files, as described in [Step 1: Create Your Script](#).

Bundle installation scripts support the entire SuiteScript API, including error handling and the debugger. For details specific to bundle installation scripts, see [Bundle Installation Script Functions](#).

#### To create a bundle installation script file:

1. Create a .js script file and add code.

This single script file should include Before Install, After Install, Before Update, After Update, and Before Uninstall functions as necessary. It can include calls to functions in other files, but you will need to list these files as library script files on the NetSuite script record.

### Add the Bundle Installation Script File to the File Cabinet

After you have created a .js file with your bundle installation script code, you need to add this file to the NetSuite file cabinet.

The following steps describe how to add the file manually. If you are using the SuiteCloud IDE, this process is automated. For more information, see [Step 2: Add Script to NetSuite File Cabinet](#).

#### To add a bundle installation script file to the file cabinet:

1. Go to Documents > Files > File Cabinet, and select the folder where you want to add the file.



It is recommended that you add your file to the SuiteScripts folder, but it can be added to any other folder of your choice.

2. Click **Add File**, and browse to the .js file.
3. In the file cabinet folder where you added the bundle installation script file, click the **Edit** link next to file.
4. Check the **Available for SuiteBundles** box.
5. Optionally, you can check the **Hide in SuiteBundle** box.  
Because this script file will be included in the bundle, by default its contents will be accessible to users of target accounts where the bundle is installed. If you do not want these users to see this file, you can set this option to hide it.
6. Click **Save**.

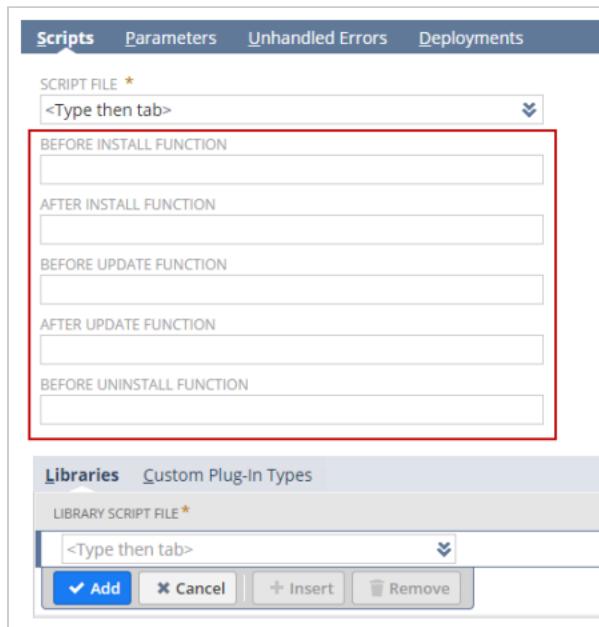
## Create the Bundle Installation Script Record

After you have added a bundle installation script file to the file cabinet, you can create a NetSuite script record.

### To create a bundle installation script record:

1. Go to Setup > Customization > Scripts > New, and click **Bundle Installation**.
2. Complete fields in the script record and save.

Although you do not need to set every field on the Script record, at a minimum you must provide a **Name** for the Script record, load your SuiteScript file to the record, and specify at least one of the executing functions on the Scripts tab.



These functions should all be in the main script file. If these functions call functions in other script files, you need to list those files as library script files.

For more details about creating a script record, see [Steps for Creating a Script Record](#).

## Define Bundle Installation Script Deployment

After you have created a bundle installation script record, you need to define at least one deployment. For details about defining script deployments, see [Step 5: Define Script Deployment](#) and [Steps for Defining a Script Deployment](#)

You can define multiple deployments per bundle installation script . When you associate the script with a bundle, you actually choose a specific deployment.

### To define a bundle installation script deployment.

1. Do one of the following:
  - When you save your Script record, you can immediately create a Script Deployment record by selecting **Save and Deploy** from the Script record **Save** button.
  - If you clicked **Save**, immediately afterwards you can click **Deploy Script** on the script record.
  - If you want to update a deployment that already exists, go to Customization > Scripting > Script Deployments. Click **Edit** next to the deployment record you want to edit.

2. Complete fields in the script deployment record and click **Save**.

Be sure to check the **Execute as Admin** box.

If you want to debug the script, set the **Status to Testing**. To enable the script to be run in a target account, you must set the **Status to Released**.

## Associate the Script with a Bundle

After a bundle installation script has been created and at least one deployment has been defined for it, you can associate the script with bundles as desired.

**Note:** The SuiteBundler feature must be enabled in your account for you to have access to the Bundle Builder where this task is completed.

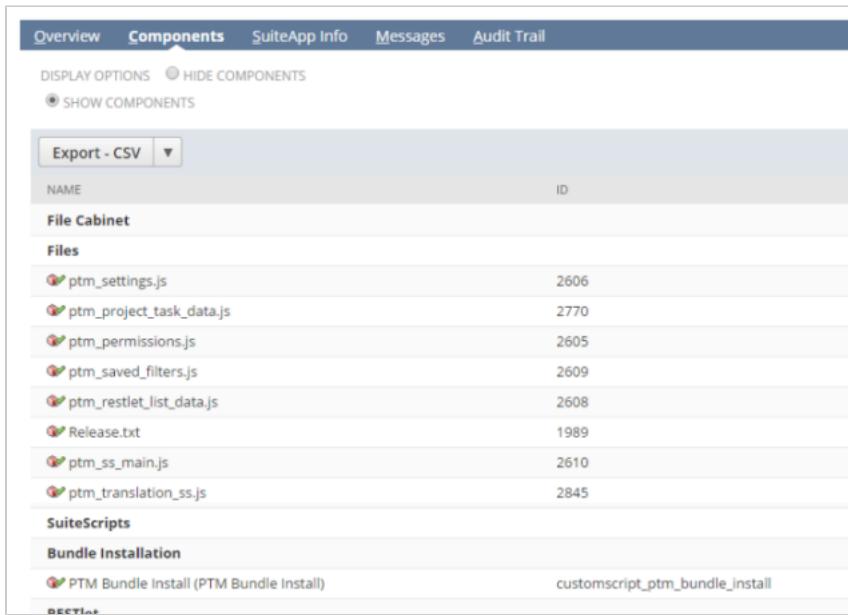
When you associate a script with a bundle, you select a specific script deployment.

### To associate a script with a bundle:

1. Start the Bundle Builder.
  - If you are creating a new bundle, go to Setup > Customization > Create Bundle.
  - If you are editing an existing bundle, go to Customization > SuiteBundler > Create Bundle > List , and select **Edit** from the Action menu for the desired bundle.
  - For details about using the Bundle Builder, see *Using the Bundle Builder*.
2. On the Bundle Basics page, select a bundle installation script deployment from the **Installation Script** dropdown.
3. Proceed through the remaining Bundle Builder steps, making definitions as necessary, and click **Save**. Note the following:
  - On the Select Objects page of the Bundle Builder, you do not have to explicitly add the bundle installation script. This script record and the related .js file are included automatically in the bundle, as are any other .js files that are listed as library script files on the script record.
  - For detailed instructions to complete all Bundle Builder steps, see *Steps for Creating a Bundle*.



After the bundle has been saved, this script record and related file(s) are listed as Bundle Components on the Bundle Details page.



NAME	ID
<b>File Cabinet</b>	
<b>Files</b>	
ptm_settings.js	2606
ptm_project_task_data.js	2770
ptm_permissions.js	2605
ptm_saved_filters.js	2609
ptm_restlet_list_data.js	2608
Release.txt	1989
ptm_ss_main.js	2610
ptm_translation_ss.js	2845
<b>SuiteScripts</b>	
<b>Bundle Installation</b>	
PTM Bundle Install (PTM Bundle Install)	customscript_ptm_bundle_install
<b>BEST PRACTICES</b>	

## Sample Bundle Installation Script

This sample includes a bundle installation script file and a library script file. For details, see the following:

- [Summary of Sample Script Files](#)
- [Sample Bundle Installation Script File Code](#)
- [Sample Library Script File Code](#)

### Summary of Sample Script Files

The bundle installation script file includes the following:

- Function that executes before bundle installation, ensuring that the Work Orders feature is enabled in the target NetSuite account, and if the bundle that is being installed is version 2.0, also ensuring that the Multiple Currencies feature is enabled
- Function that executes after bundle installation, creating an account record in the target account (note that accounts are not available to be included in bundles)
- Function that executes before bundle update, ensuring that the Work Orders feature is enabled in the target NetSuite account, and if the target account bundle is being updated to version 2.0, also ensuring that the Multiple Currencies feature is enabled
- Function that executes after bundle update, creating an account record in the target account if the update changed the bundle version number

The library script file includes a function that is called by the bundle installation script functions executed before installation and before update.

- This function checks whether a specified feature is enabled in the target account and returns an error if the feature is not enabled.
- When an error is returned, bundle installation or update terminates.

## Sample Bundle Installation Script File Code

The bundle installation script file **SampBundInst.js** contains the following code.

```

function beforeInstall(toversion)
{
    // Always check that Workorders is enabled
    checkFeatureEnabled('WORKORDERS');

    // Check that Multi Currency is enabled if version 2.0 is being installed
    if (toversion.toString() == "2.0")
        checkFeatureEnabled('MULTICURRENCY');
}

function afterInstall(toversion)
{
    // Create an account record
    var randomnumber=Math.floor(Math.random()*10000);
    var objRecord = nlapiCreateRecord('account');
    objRecord.setFieldValue('accttype','Bank');
    objRecord.setFieldValue('acctnumber',randomnumber);
    objRecord.setFieldValue('acctname','Acct '+toversion);
    nlapiSubmitRecord(objRecord, true);
}

function beforeUpdate(fromversion, toversion)
{
    // Always check that Workorders is enabled
    checkFeatureEnabled('WORKORDERS');
    // Check that Multi Currency is enabled if version 2.0 is being installed
    if (toversion.toString() == "2.0")
        checkFeatureEnabled('MULTICURRENCY');
}

function afterUpdate(fromversion, toversion)
{
    // Do not create an account if updating with the same version as the one installed
    if (fromversion.toString() != toversion.toString())
    {
        // Create an account record
        var randomnumber=Math.floor(Math.random()*10000);
        var objRecord = nlapiCreateRecord('account');
        objRecord.setFieldValue('accttype','Bank');
        objRecord.setFieldValue('acctnumber',randomnumber);
        objRecord.setFieldValue('acctname','Acct '+toversion);
        nlapiSubmitRecord(objRecord, true);
    }
}

```

```
}
```

## Sample Library Script File Code

The library script file **CheckFeat.js** contains the following code.

```
function checkFeatureEnabled(featureId)
{
    nlapiLogExecution('DEBUG','Checking Feature',featureId);
    var objContext = nlapiGetContext();
    var feature = objContext.getFeature(featureId);

    if ( feature )
    {
        nlapiLogExecution('DEBUG','Feature',featureId+' enabled');
    }

    else
    {
        throw new nlobjError('INSTALLATION_ERROR','Feature '+featureId+' must be enabled. Please enable the feature and re-try.');
    }
}
```



# Setting Runtime Options Overview



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

If you have not already created a Script Deployment record for your SuiteScript file, please see [Step 5: Define Script Deployment](#). This is the last step that is required for you to run your script in NetSuite.

If you have created a deployment for your script and would now like to set additional deployment options, see the following topics. These topics do not need to be read in order.

- [Setting Script Execution Event Type from the UI](#)
- [Setting Script Execution Log Levels](#)
- [Executing Scripts Using a Specific Role](#)
- [Setting Available Without Login](#)
- [Setting Script Deployment Status](#)
- [Defining Script Audience](#)
- [Creating Script Parameters Overview](#)

Also see these topics for information related to script deployments, but not necessarily specific to any deployment/runtime options.

- [Creating a Custom Script Deployment ID](#)
- [Viewing Script Deployments](#)
- [Creating Script Execution Logs](#)

# Setting Script Execution Event Type from the UI



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the **Event Type** field on the Script Deployment page, you can specify the event type that you want to trigger the execution of the script (see figure). If Event Type is left blank, the script will execute only on the events specified in the actual .js script file.

The screenshot shows the 'Script Deployment' page for a script named 'CreateFollowUpTask'. The 'APPLIES TO' field is set to 'Opportunity'. The 'Status' is 'Testing'. The 'Event Type' dropdown menu is open, showing options: Edit (selected), Edit Forecast, Email, Mark Complete, Order Items, Pack, and Pay Bills. Below the dropdown, there are tabs for Audience, Scripts, Execution Log, and History. At the bottom, there are buttons for ROLES (with 'Select All' checked) and SUBSIDIARIES.



**Note:** The Event Type field appears on Suitelet, user event, and record-level client Script Deployment pages only.

The Event Type field is useful if you want to specify a script execution context right at the time of script deployment – without having to modify your .js file. After you specify an event type and click Save, the deployed script will execute only on that event, regardless of the event types specified in the script file.

Be aware that event types specified in the UI take precedence over the types specified in the script file. For example, if the **create** event type is specified in the script, selecting **edit** from the Event Type field will restrict the script from running on any event other than **edit**.

The following snippet is from a user event script. Notice that the event type specified in the code is **create**. If the **edit** event type is specified in the UI, this script will execute only when the specified record is edited, not created:

```
function followUpCallAfterSubmit(type)
{
    // execute the logic in this script if a new customer is created
    if (type == 'create')
    {
        // obtain a handle to the newly created customer record
        var custRec = nlapiGetNewRecord();

        // remainder of script.....
    }
}
```

# Setting Script Execution Log Levels



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the **Log Level** field on the Script Deployment page, specify which log entries you want to appear on the Execution Log tab (see figure).

The screenshot shows the 'Script Deployment' page. At the top, there are buttons for Save, Cancel, Reset, Change ID, and Actions. The 'SCRIPT' section contains 'CreateFollowUpTask'. The 'APPLIES TO' dropdown is set to 'Opportunity'. The 'ID' section shows 'customdeploy1' with a checked checkbox labeled 'DEPLOYED'. On the right, there are fields for 'STATUS \*' (set to 'Testing'), 'EVENT TYPE' (set to 'Edit'), and 'LOG LEVEL' (set to 'Debug'). A dropdown menu for 'LOG LEVEL' lists four options: Debug, Audit, Error, and Emergency. Below the main form, there is a navigation bar with tabs for Audience, Scripts, Execution Log, and History.

See [Creating Script Execution Logs](#) for details on how to further customize your view of all log entries.

The Log Level field is essentially used as a basic filtering mechanism. Each log entry written with `nlapilogExecution(type, title, details)` specifies a log level in the `type` argument. When logging details are displayed on the Execution Log tab of the Script Deployment record, you can specify that all messages will display (by selecting Debug from the Log Level field) or a specific type of message.



**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 [Governance on Script Logging](#).

Use the Log Level field to filter which messages will appear on the Execution Log tab of the Script Deployment page. Set filtering by selecting any of the following log levels from the Log Level Field:

- **Debug** : For scripts in testing mode. A log level set to Debug shows all Audit, Error, and Emergency information on the Execution Log tab.
- **Audit** : For scripts going into production. A log level set to Audit provides a record of events that have occurred during the processing of the script (for example, "A request was made to an external site.").
- **Error** : For scripts going into production. A log level set to Error shows only unexpected script errors.
- **Emergency** : For scripts going into production. A log level set to Emergency shows only the most critical errors in the script log.



**Note:** The log level you specify in the UI is independent of any error handling within your script.

# Executing Scripts Using a Specific Role



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the **Execute as Role** field on the Script Deployment page, select the role you want the script to run as. The Execute as Role field provides role-based granularity in terms of the permissions and restrictions of the executing script. In the figure below, the script will always execute based on the permissions and restrictions assigned to the Sales Person role, even if the role of the logged in user is different.

Note that even if the logged in user's role is Admin, when the script executes in this user's account, it will execute based on the record-level permissions assigned to the Sales Person role.

The screenshot shows the 'Script Deployment' page. At the top, there are buttons for 'Save', 'Cancel', 'Reset', 'Change ID', and 'Actions'. Below these are fields for 'SCRIPT' (AddSubTabtoForm), 'APPLIES TO' (Sales Order), 'ID' (customdeploy1), and a checked checkbox for 'DEPLOYED'. On the right side, there are dropdowns for 'STATUS' (Released), 'EVENT TYPE', 'LOG LEVEL', and 'EXECUTE AS ROLE'. The 'EXECUTE AS ROLE' dropdown is set to 'EP Sales Person' and is highlighted with a red border.

SuiteScript developers can also select from custom roles that they have created with permissions that are specific to a script deployment.

Also note that the value of **Current Role** in the Execute as Role field means that the script will execute using the permissions of the currently logged-in user (the user whose account the script is running in).



**Note:** Be aware that when a script triggers other scripts, the cascading scripts will run as the role of the initial triggering script's role, not the role specified on the cascaded script's role.

The **Execute as Role** field appears on the Script Deployment pages for these script types:

- Suitelet
- User Event
- Portlet
- Mass Update
- Workflow Action



**Note:** For information about role restrictions in client SuiteScript, see [Role Restrictions in Client SuiteScript](#).



**Note:** Also, only users with the administrator role have access to the Execute as Role field. This field is disabled for all other users who may be working with SuiteScript. Even if you have been granted "Full" access to SuiteScript (on the permissions tab of your NetSuite account), unless your role is also an administrator, you will be unable to change the setting of the Execute As Role field. This is also true for the Execute as Admin check box that appears on the summary page in SuiteFlow.

## Version 2013 Release 1 Behavior Change for Scripts Run as Administrator

In Version 2013 Release 1, the Execute as Role field replaced the Execute as Admin check box. Prior to this release, NetSuite administrators and SuiteScript developers had to use the Execute as Admin check box to ensure that their scripts executed as intended, without the user who was executing the script receiving permission errors.

As part of this change, this release implemented a change in behavior for scripts run as administrator. Prior to Version 2013 Release 1, scripts with the Execute as Admin option enabled ran with administrator permissions, but with some exceptions. For example, these scripts did not have the Override Period Restrictions or Allow Non G-L Changes permissions. Now, if you select Administrator in the Execute as Role field for a script, the script runs with ALL permissions. So, a script that did not post transactions in a locked period when executed as admin before this change, now will post transactions in a locked period if Administrator is selected for Execute as Role.

To adapt to this behavior change, you may need to select roles other than administrator for some scripts that previously had the Execute as Admin option enabled. If no current roles have the range of permissions needed, you can create new roles as needed and select them in the Execute as Role field.

## Testing a Script Using Different Roles

NetSuite allows you to switch the deployment status to Testing when you are setting the script to execute as a different role. In this scenario, the script will be triggered only by you, but when it executes, it will execute as the role you selected.

## Setting Your Script to Run With Administrative Privileges

If you want the script to execute using administrative privileges, regardless of the permissions of the currently logged-in user, select Administrator in the Execute as Role field.

Sometimes there are scripts which may require that they run with administrative privilege. For example, if you have a script that creates follow-up tasks after a sales order has been saved, and the script needs to read data from employee records, the script will not complete execution if a user's role does not have permission to access employee records. In this case, it may be appropriate to have the script set to Administrator in the Execute as Role field.



**Note:** All bundle installation scripts need to execute as Administrator, so this option should always be enabled for this type of script deployment.

However, setting a script to execute as Administrator should be considered carefully, as this option allows scripts to execute with privileges that the logged in user does not have. This may be appropriate



for certain scripts, but there are other cases where the script performs actions that are only appropriate for certain roles.



**Note:** Often when scripts execute without logins or in the Web store, they tend to be implemented as an Administrator whenever any meaningful interaction with the system is required.

# Setting Available Without Login



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the **Available Without Login** check box on the Script Deployment page, select the check box to allow users without an active NetSuite session to have access to the Suitelet.



**Note:** The Available Without Login check box appears on the Script Deployment page for Suitelets only.

Script Deployment		Actions	
SCRIPT Simple Suitelet Form	STATUS Released	<input checked="" type="checkbox"/> AVAILABLE WITHOUT LOGIN URL <code>/app/site/hosting/scriptlet.nl?script=48&amp;deploy=1</code>	
TITLE Simple Suitelet Form	EVENT TYPE		
ID customdeploy1	LOG LEVEL Error	EXTERNAL URL <code>https://forms.netsuite.com/app/site/hosting/scriptlet.nl?script=48&amp;deploy=1&amp;compid=1013519&amp;h=4444e73b825e845db181</code>	
<input checked="" type="checkbox"/> DEPLOYED		EXECUTE AS ROLE Administrator	

When you select Available Without Login and then save the Script Deployment record, an External URL appears on the Script Deployment page (see figure above). Use this URL for Suitelets you want to make available to users who do not have an active NetSuite session.

The following are a few uses cases that address when you might want to make a Suitelet externally available:

- hosting one-off online forms (capturing test drive sign-up requests or partner conference registrations, for example)
- inbound partner communication (such as - listening for payment notification responses from PayPal or Google checkout, or for generating the unsubscribe from email campaigns page, which requires access to account information but should not require a login or hosted website)
- for Facebook/Google/Yahoo mashups in which the Suitelet lives in those web sites but needs to communicate to NetSuite via POST requests



**Note:** Because there are no login requirements for Suitelets that are available without login, be aware that the data contained within the Suitelet will be less secure.

## Errors Related to the Available Without Login URL

Based on the use case for your Suitelet, you will use either the internal URL or the external URL as the launching point for the Suitelet.

Some of the factors determining whether the Suitelet will deploy successfully are the dependencies between the type of URL you are referencing (internal or external), the Suitelet deployment status (Testing or Released), and whether the **Select All** check box has been selected on the Audience tab of the Script Deployment page. The following table summarizes these dependencies.

Suitelet URL Type	Deployment Status	Select All check box	Result
internal	Testing	not checked	Suitelet deploys successfully
internal	Testing	checked	Suitelet deploys successfully
internal	Released	not checked	Error message: You do not have privileges to view this page.
internal	Released	checked	Suitelet deploys successfully
external	Testing	not checked	Error message: You are not allowed to navigate directly to this page.
external	Testing	checked	Error message: You are not allowed to navigate directly to this page.
external	Released	checked	Suitelet deploys successfully
external	Released	not checked	Error message: You do not have privileges to view this page.

# Setting Script Deployment Status

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the **Status** field on the Script Deployment page, set the deployment status of the script to either Testing or Released.

**Note:** The Testing and Released statuses do not apply to scheduled scripts. To learn about scheduled script deployment statuses, see [Understanding Scheduled Script Deployment Statuses](#) or [Executing a Scheduled Script in Certain Contexts](#).

The screenshot shows the 'Script Deployment' page for a script named 'CreateFollowUpTask'. The 'APPLIES TO' dropdown is set to 'Opportunity'. The 'ID' field contains 'customdeploy1' and the 'DEPLOYED' checkbox is checked. On the right side, the 'STATUS' dropdown is set to 'Testing' and is highlighted with a red box. Other settings shown include 'EVENT TYPE' (dropdown), 'LOG LEVEL' (set to 'Debug'), and 'EXECUTE AS ROLE' (set to 'Current Role'). Below the main form, the 'Audience' tab is selected, displaying audience categories like ROLES, DEPARTMENTS, SUBSIDIARIES, GROUPS, EMPLOYEES, and PARTNERS, each with a 'Select All' checkbox and a scrollable list of items.

## Status Set to Testing

When a script's deployment status is set to Testing, the script will execute for the script owner **only**. Even if you have defined an audience on the Audience tab and saved the Script Deployment record, the script will still only run for the script owner so long as it is in testing mode.

Note that script owners can use the Audience tab, along with the Testing status, to test scripts for various audience types. For information, see [Using the Audience Tab to Test Scripts](#).

Also note that when using the SuiteScript Debugger to test scripts, the script's deployment status must be set to Testing. You cannot debug a Deployed script if the status has been set to Released. (See [Deployed Debugging](#) for more information on using the SuiteScript Debugger to test existing scripts.)

If you are working with Suitelet Script Deployment records, also see [Errors Related to the Available Without Login URL](#). This section discusses the relevance of the Testing status as it pertains to internally and externally available Suitelets.



**Note:** A bundle installation script cannot execute in target accounts if its deployment status is set to Testing.

## Status Set to Released

A script deployment status set to Released means that the script will run in the accounts of all specified audience members. (See [Defining Script Audience](#) for information on defining script audiences.) When the deployment status is set to Released, the script is considered to be “production ready.”

Be aware that if you do not specify any values on the Audience tab, the script will execute for no one other than the script author/owner, even if the script deployment status is set to Released.



**Note:** Bundle installation scripts do not have an Audience. If the deployment status is set to Released for this type of script, it executes automatically in target accounts when the associated bundle is installed or updated.

If you are working with Suitelet Script Deployment records, also see [Errors Related to the Available Without Login URL](#). This section discusses the relevance of the Released status as it pertains to internally and externally available Suitelets.

# Defining Script Audience

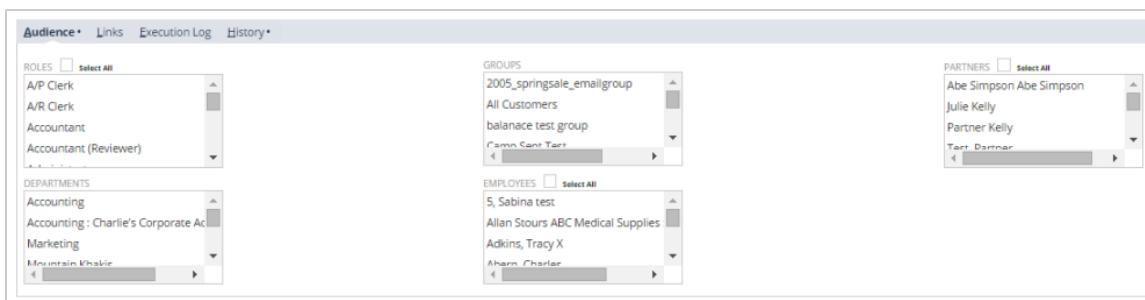


**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

On the **Audience** tab on the Script Deployment page, define the audience access levels for the script. When the script is deployed, it will run in the accounts of only the specified audiences.



**Note:** The Audience tab appears on the Script Deployment page for these script types: Suitelet, portlet, user event, global client, action. You cannot specify an audience for scheduled scripts or bundle installation scripts.



If you do not specify any values on the Audience tab, the script will execute for no one other than the script author/owner, even if the script deployment status is set to Released. (For information on the differences between the Released and Testing deployment statuses, see [Setting Script Deployment Status](#).)

If you choose both role and department options, a user must belong to one of the selected roles AND one of the selected departments to access the search. If you choose options for any other combinations of types, a user need only belong to a selected option of one type OR of another.

If you want the script to run in the accounts of all NetSuite users, select the **Select All** check box next to Roles. If you want the script to run in the accounts of only specific users, select the appropriate roles, departments, groups, employees, or partners.

Be sure to save the Script Deployment record after all audience members have been defined.

Important Things to Note:

- If you are working with Suitelet Script Deployment records, also see [Errors Related to the Available Without Login URL](#). This section discusses the relevance of the **Select All** check box as it pertains to internally and externally available Suitelets.
- Mass update script deployments and mass updates can both be assigned an audience. It is the users responsibility to make sure the two audiences are in sync. For information about working with the mass update script type, see [Mass Update Scripts](#).

## Using the Audience Tab to Test Scripts

If the Status field on the Script Deployment record is set to **Testing**, you can test scripts assigned to specific audiences as long as you are a member of that audience type. (For information on the Testing deployment status, see [Setting Script Deployment Status](#).)

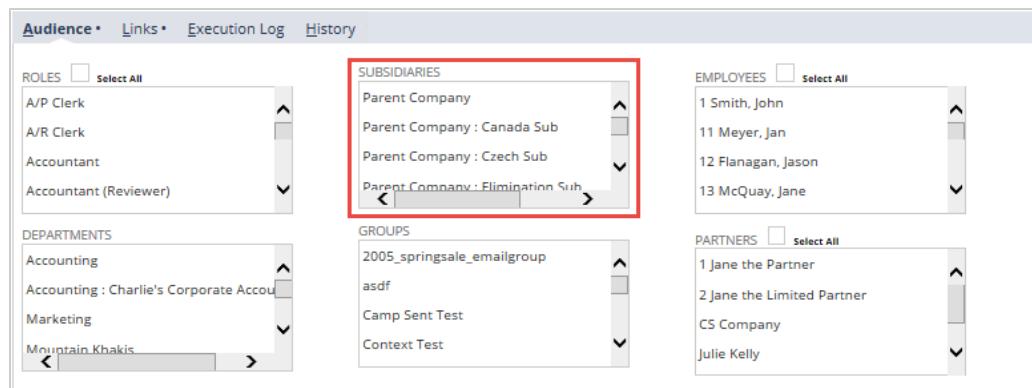
For example, if you have written a script that you want to run for everyone in the Support Management role, you can log into NetSuite and then switch to the Support Management role to test the script. As long as the deployment status for this script is set to Testing, the script will run for no one other than you (as the script owner). After you have determined that the script runs as expected for the Support Management role, you can change the script's deployment status to Released. After the deployment is set to Released, it will run in the account for all those assigned to the Support Management role.

This is a good approach for script owners to verify that their script will run in the accounts for specified roles, departments, groups, employees, or partners.

## Using the Audience Tab in OneWorld Accounts

Script authors/owners who are developing scripts for NetSuite OneWorld accounts can specify script audience based on subsidiary. In OneWorld accounts, the Audience tab includes a Subsidiaries multiselect field. After choosing subsidiaries, be sure to click Save on the Script Deployment page.

Script owners working in a OneWorld account that has multiple subsidiaries can select the subsidiaries they want their script to run in, and then log into an account of one of the specified subsidiaries. As long as the script deployment status is set to Testing, the script will not run for any of the subsidiary employees other than the script owner. This is a good way for script owners to verify that the script will run in accounts for specified subsidiaries.



# Creating Script Parameters Overview

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In the context of SuiteScript, script parameters are essentially **custom fields**. Script parameters are not considered to be parameters that are passed between JavaScript functions.

A script parameter can have any of the characteristics of a custom field created through point-and-click customization. They are configurable by administrators and end users and are accessible programmatically through SuiteScript.

Script parameters are defined on the Parameters tab of the Script record page.

The following topics are covered in this section. They do not need to be read in order, although it is recommended if you are not familiar with script parameters.

- Why Create Script Parameters?
- Creating Script Parameters
- Referencing Script Parameters
- Setting Script Parameter Preferences

**Note:** If you do not have experience with NetSuite custom fields, it is recommended that you review [Creating a Custom Field](#) in the NetSuite Help Center.

# Why Create Script Parameters?

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You should create script parameters if you want one or more parts of your program to be configurable, either through script deployment or by the end user. Script parameters should be created whenever you need to parameterize a script that was deployed multiple times. This makes it easy to customize the behavior of the program for **each** deployment.

**Note:** You can also configure scheduled scripts by specifying the configuration parameters as arguments to `nlapiScheduleScript(scriptId, deployId, params)`.

Deployment-specific parameters allow you to configure program behavior without having to write code. This is particularly useful when admins deploy scripts that were installed as part of a bundle. This allows them to control/modify the program without having to know anything about the code. In other words, this is like the properties/config file that most applications have which allow you to modify the runtime behavior.

Having script parameters also allows you to modify program behavior for troubleshooting purposes without having to change code, which is often expensive and infeasible (for example, when the script writer is unavailable). Finally, script parameters give you the flexibility of being able to handle a wide range of inputs depending on the context (for example, one script deployed to 50 different records, each requiring a slightly different behavior). The alternative would be to hard-code and deploy 50 different scripts. The downside to the latter is additional maintenance (the code is not configurable, potential code duplication, changes in business requirements require code changes).

**Note:** You do not need to create script parameters if your program is not meant to be configurable, in other words, everything is hard-coded.



# Creating Script Parameters

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Use the following steps to create script parameters. If you are unsure how to create a Script record, see [Step 4: Create Script Record](#).

1. On the Script record, click the Parameters tab.

**Note:** If you want to add a parameter to a Script record that already exists, go to Customization > Scripts > [script], where [script] is the desired script record. Open the Script record in Edit mode. Click the Parameters tab and then click the **New Parameter** button that appears on the tab.

2. In the Label field, type the name of the parameter (custom field) as it will appear in the UI once the script is deployed. The figure below shows that the field label on the UI will appear as **Check Box Required**.
3. In the ID field, create a custom ID for the script parameter. You can also leave the ID field blank and accept a system-generated ID. (It is a best practice to create your own ID for the script parameter. Doing so will help avoid naming conflicts should you later decide to bundle your script.)

**Note:** Script parameter IDs must be in lowercase and contain no spaces. Also note that parameter IDs cannot exceed 30 characters.

4. From the Type drop-down list, select the parameter's field type (for example, Hyperlink, Date, Free-Form Text, or in this case, Check Box). For more information on field types, see the help topic [Table of Custom Field Type Descriptions](#) in the NetSuite Help Center.

**Note:** If the parameter's type is List/Record, specify the list or record using the List/Record drop-down list (see figure below).

DESCRIPTION	ID	TYPE	LIST/RECORD	PREFERENCE
Check Box Required	custscript_checkbox1	Check Box		Company

Also note that if you define a **saved search** as a List/Record script parameter, only saved searches that are public will appear in the List/Record parameter dropdown field. For more information on working with searches using SuiteScript, see [Searching Overview](#) in the NetSuite Help Center.

5. From the Preference dropdown, set the script parameter preference to Company, User, or Portlet (if creating a portlet script). Based on the preference, the parameter will default



to the values set on either the General Preferences page, the Set Preferences page, or the portlet setup page. If no preference is specified, the script parameter is considered to be a "deployment script parameter," and its value is defined on the Script Deployment record.

For more information setting parameter preferences, see [Setting Script Parameter Preferences](#).

6. Once you have defined script parameter properties, click Add.
7. If you have finished defining the parameter, save the Script record and set the script's deployment values on the Script Deployment page.

In this example, when the "Simple Form" Suitelet is deployed, the check box script parameter appears on the form.

8. To define additional properties for the script parameter, such as display size, validation, or sourcing values, save the Script record and then re-open the Script record.
9. On the Script record page, click the Parameters tab, and then click the link to the script parameter you originally created.

After clicking the parameter link, the Script Field page opens. Use this page to define additional values for the script parameter. For information on setting display, validation, and sourcing values, see these sections in the NetSuite Help Center:

- [Setting Display Options for Custom Fields](#)
- [Setting Validation and Defaulting Properties](#)
- [Setting Sourcing Criteria](#)
- [Setting Filtering Criteria](#)

10. Click Save on the Script Field page after defining all values.

# Referencing Script Parameters

In your SuiteScript code you must use the `nlobContext.getSetting(type, name)` method to reference script parameters. For example, to obtain the value of a script parameter called `custscript_checkboxtest2`, you must use the following code:

```
//Add to the Suitelet a check box script parameter called Check Box Required  
var field = form.addField('custscript_checkboxtest2', 'checkbox', 'Check Box Required');  
  
//Get the parameter context and return a checked check box if value  
//is set to T. Note that check box values are set to T or F, not true or false  
  
if (nlapiGetContext().getSetting('SCRIPT', 'custscript_checkboxtest2') == 'T' {  
    field.setDefaultValue('T')  
}  
  
...remainder of Suitelet code
```

Be aware that you cannot write to a script parameter using SuiteScript. Although you can read from these fields, you cannot write to them. The only time you can pass a value to a script parameter outside of the UI is when you call `nlapiScheduleScript(scriptId, deployId, params)`. In the API documentation for this function, see [Example 5 - Passing Script Parameters in a Scheduled Script](#).



# Setting Script Parameter Preferences



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

As a script author, NetSuite gives you the ability to specify the preference type for each script parameter (see figure). Available preference types are:

- **Company:** If the parameter preference is set to Company, the script parameter's value is read from the value specified in Setup > Company > General Preferences > Custom Preferences tab. See the [Example](#) later in this section.
  - **User:** If the preference is set to User, the parameter's value is read from the value set in Home > Set Preference > Custom Preferences tab.
- Here, end users can override the default (company) script behavior and insert their own default value. End users do not have to manipulate a script or its deployments to change or customize the parameter.
- <blank>: If you do not set a preference, the script parameter is considered a “deployment” script parameter by default. In this case, you will define the value of the script parameter on the Parameters tab of the Script Deployment record .



**Note:** See [Creating Script Parameters](#) for steps on creating a script parameter. Also see [Referencing Script Parameters](#) for information on accessing script parameter values.

LABEL *	ID	TYPE	LIST/RECORD	PREFERENCE
Check Box Required	_checkbox2	Check Box		Company

At the time these three script parameters were created, no preferences were set. In this case, parameter values are defined on the Parameters tab of the Script Deployment record.

**Script Deployment**

Save ▾ Cancel Reset

SCRIPT  
Simple Suitelet Form

TITLE \*  
Simple Suitelet Form

ID

DEPLOYED

STATUS \*  
Testing

EVENT TYPE  
LOG LEVEL  
EXECUTE AS ROLE  
 AVAILABLE WITHOUT LOGIN

Audience • Links Parameters • Execution Log

CHECK BOX REQUIRED  
MY COMPANY'S EMAIL  
MY SCRIPT PARAMETER FIELD

Note that users who install a bundled script that uses preferences can override the default behavior of the script and customize the script to their specific business needs. Setting preferences eliminates having to manipulate the script code or the script deployment. (For information on bundling scripts, see the *SuiteBundler Overview* topic in the NetSuite Help Center.)

### Example

In this example, the parameter called **Check Box Required** (with the internal ID `custscript_checkboxtest2`) is set to the Company preference.

**Script**

Edit Back Deploy Script Actions ▾

TYPE Suitelet	DESCRIPTION This Suitelet creates a simple form that includes a check box.
NAME Simple Suitelet Form	OWNER K Wolfe
ID customscript48	<input type="checkbox"/> INACTIVE

Scripts Parameters Unhandled Errors Execution Log Deployments History

New Parameter

DESCRIPTION	ID	TYPE	LIST/RECORD	PREFERENCE
Check Box Required	custscript_checkbox1	Check Box		Company

By going to Setup > Company > General Preferences > Custom Preferences tab (see below), administrators can set the default value of this parameter for the entire company. In this example the value of the **Check Box Required** script parameter is set to T (the check box is checked).

Overriding Preferences • Languages • Centers Custom Preferences •

General NetS

CHECK BOX REQUIRED OUTI

When the Suitelet that contains this check box is deployed, the **Check Box Required** script parameter will appear checked.

If the **Check Box Required** parameter had been set to F (the check box contained no check mark), the check box would have appeared empty on the form when the Suitelet was deployed.

## Script Parameter Preferences and Bundles

Bundled script parameters that have a user or company preference set are not updated in target accounts when the bundle is updated. However, script parameters that do not have a preference specified are considered part of the script deployment, and whether they are updated in target accounts when the bundle is updated depends on the setting of the related bundle object preference:

- If the preference for the bundled script is set to Update Deployments, script deployment parameters are updated in target accounts to match those in the source account.
- If the preference for the bundled script is set to Do Not Update Deployments, script deployment parameters are not updated in target accounts.

If bundle authors expect target account users to want to change parameter values for a bundled script, on the script record they should set the Preference for these parameters to be either Company or User. Target account users can then change parameter values as needed, and these values are not affected on bundle update, even if the related bundle object preference is set to Update Deployments.

To prevent changes to target account script deployment parameters that do not have a preference set, set the related bundle object preference to Do Not Update Deployments.

For more information about bundle object preferences, see the help topic [Setting Bundle Object Preferences](#).



# Searching Overview

Similar to much of the searching functionality available through the NetSuite UI, SuiteScript [Search APIs](#) allow you to retrieve real-time data from your account. You can search for a single record by keywords, create saved searches, search for duplicate records, or return a set of records that match filters you define.

The following sections provide details on searching with SuiteScript. If you are new to SuiteScript searches, it is recommended that you read these topics in order.

- [Understanding SuiteScript Search Objects](#)
- [Search Samples](#)
- [Search APIs](#)
- [Supported Search Operators, Summary Types, and Date Filters](#)

# Understanding SuiteScript Search Objects

The basis of most SuiteScript searches use the following objects:

1. `nlobjSearchFilter` - used to define filtering criteria for the search
2. `nlobjSearchColumn(name, join, summary)` - used to define search return columns for the search
3. `nlobjSearchResult` - used to get the values of specific search results

After all filters and search columns are defined, the search is executed using the `nlapiSearchRecord(type, id, filters, columns)` function.

**Important:** If you are performing a global search or a duplicate record search, you will **not** use the objects listed above or the `nlapiSearchRecord(...)` function. For details on these types of searches, see [Searching for Duplicate Records](#) and [Performing Global Searches](#).

## Defining Search Filters

The following figure shows the UI equivalent of using the `nlobjSearchFilter` object to define search filters. In the UI, users define search filters by clicking the Criteria tab on the search record (in this case the Customer Search record).

In SuiteScript, the same filter value is specified through the `nlobjSearchFilter` object:

```
var filters = new Array();
filters[0] = new nlobjSearchFilter ('companynamne', null, 'startswith', 'A' );
```

The screenshot shows the 'Customer Search' interface. At the top, there are buttons for 'Submit', 'Reset', 'Export', 'Personalize Search', and 'Create Saved Search'. Below these is a checkbox for 'USE ADVANCED SEARCH' which is checked. The main area has two tabs: 'Criteria' (which is selected and highlighted in red) and 'Results'. A note below the tabs says 'Use this tab to specify criteria that narrow down your search.' There is also an unchecked checkbox for 'USE EXPRESSIONS'. The 'Criteria' tab displays a table with a single row. The first column is 'FILTER \*' and the second is 'DESCRIPTION \*'. The row contains 'Company Name' and 'startswith A'. At the bottom of the table are buttons for 'Add', 'Cancel', 'Insert', and 'Remove'. At the very bottom of the search interface are the same 'Submit', 'Reset', 'Export', 'Personalize Search', and 'Create Saved Search' buttons.

## How do I know which search filters I can use in my code?

To figure out which search filters are available for a specific record type:

1. See the section [SuiteScript Supported Records](#) in the NetSuite Help Center.
2. Click on the record you are running your script against.
3. In the documentation for that record, see the **Search Filters** table. All available search filters for that record type will be listed in the table.

## Defining Search Columns

The following figure shows the UI equivalent of using the `nlobjSearchColumn(name, join, summary)` object to define search return columns. In the UI, users define search columns by clicking the Results tab on the search record (in this case the Customer Search record).

In SuiteScript, the same column values are specified using:

```
var columns = new Array();
columns[0] = new nlobjSearchColumn('entity');
columns[1] = new nlobjSearchColumn('phone');
columns[2] = new nlobjSearchColumn('companyname');
```

The screenshot shows the 'Customer Search' interface. At the top, there are buttons for 'Submit', 'Reset', 'Export', 'Personalize Search', and 'Create Saved Search'. Below these is a checked checkbox for 'USE ADVANCED SEARCH'. The main area has two tabs: 'Criteria' (selected) and 'Results' (highlighted with a red border). A note says 'Use this tab to indicate columns to be included in the search results as well as sort order.' Below this are sorting options: 'SORT BY' dropdown set to 'ID', 'DESCENDING' checkbox; 'THEN BY' dropdowns with 'DESCENDING' checkboxes; and 'OUTPUT TYPE' dropdown set to 'Normal', 'SHOW TOTALS' checkbox. At the bottom of the search area is a table titled 'FIELD \*' with three rows: 'Company Name', 'Phone', and 'Name'. The first two rows are highlighted with a red box.

FIELD *	SUMMARY TYPE	FUNCTION	FORMULA	WHEN ORDERED BY FIELD	CUSTOM LABEL	CUSTOM LABEL 1
Company Name						
Phone						
Name						

## How do I know which search columns I can use in my code?

To figure out which search columns are available for a specific record type:

1. See the section [SuiteScript Supported Records](#) in the NetSuite Help Center.
2. Click on the record you are running your script against.
3. In the documentation for that record, see the **Search Columns** table. All available search columns for that record type will be listed in the table.

## Executing the Search

In the UI, users click the Submit button to execute a search. In SuiteScript, the equivalent of clicking the Submit button is calling the `nlapiSearchRecord(type, id, filters, columns)` function:

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter ('companyname', null, 'startswith', 'A');

// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn('entity');
columns[1] = new nlobjSearchColumn('phone');
```

```
columns[2] = new nlobjSearchColumn('companyname');
// Execute the Customer search. You must specify the internal ID of
// the record type you are searching against. Also, you will pass the values
// defined in the filters and columns arrays.
var searchResults = nlapiSearchRecord('customer', null, filters, columns);
```

## Getting Search Return Values

If you want to get specific values returned by the search, you will use the [nlobjSearchResult](#) object to specify the values.

```
// Define search filters
var filters = new Array();
filters[0] = new nlobjSearchFilter ('companyname', null, 'startswith', 'A');
// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn('entity');
columns[1] = new nlobjSearchColumn('phone');
columns[2] = new nlobjSearchColumn('companyname');
// Execute the search
var searchResults = nlapiSearchRecord('customer', null, filters, columns);
// Get the value of the Company Name column
var values = searchResults[0].getValue(columns[2]);
```



# Search Samples

The following are samples of SuiteScript searches.

- Creating Saved Searches
- Using Existing Saved Searches
- Filtering a Search
- Returning Specific Fields in a Search
- Searching on Custom Records
- Searching Custom Lists
- Executing Joined Searches
- Searching for an Item ID
- Searching for Duplicate Records
- Performing Global Searches
- Searching CSV Saved Imports
- Using Formulas, Special Functions, and Sorting in Search
- Using Summary Filters in Search

For more general information that describes search objects, see [Understanding SuiteScript Search Objects](#).

## Creating Saved Searches

The `nlobjSearch` object is the primary object used to encapsulate a NetSuite saved search. Note, however, you are **not required** to save the search results returned in this object.

To create a saved search, you will first define all search criteria and then execute the search using `nlapiCreateSearch(type, filters, columns)`. The search will not be saved until you call the `nlobjSearch.saveSearch` method.

By default, searches returned by `nlapiCreateSearch(...)` will be private, which follows the saved search model in the UI. To make a saved search public, you must set the `nlobjSearch.setIsPublic(type)` method to true.

### Creating a Saved Search Using Search Filter List

```
// 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');
```

## Creating a Saved Search Using Search Filter Expression

```
//Define search filter expression
var filterExpression = [ [ 'trandate', 'onOrAfter', 'daysAgo90' ],
    'or',
    [ 'projectedamount', 'between', 1000, 100000 ],
    'or',
    'not', [ 'customer.salesrep', 'anyOf', -5 ] ];

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

## Using Existing Saved Searches

NetSuite saved searches allow you to create reusable search definitions with many advanced search filters/results display options. Although saved searches must be created in the UI, you can pass the internal ID of the saved search to SuiteScript and re-execute the search on a regular basis. In this manner, you can keep the searches up-to-date for all who might need to access the results.

To re-execute an existing saved search, you will use [nlapiSearchRecord\(type, id, filters, columns\)](#). The filters and columns parameters represent the criteria and results columns that you want to be included when the search is re-executed.



**Note:** For general information on NetSuite saved searches, see the help topic [Using Saved Searches](#) in the NetSuite Help Center.

When using the `nlapiSearchRecord` function to execute an existing saved search, note the following:

- Only saved searches on record types currently supported by SuiteScript can be executed. For a list of records that support SuiteScript, see [SuiteScript Supported Records](#) in the NetSuite Help Center.
- Saved searches acted on by SuiteScript should be protected. If a saved search is edited after script deployment is complete, the execution of the script could fail. You can add security to a saved search by defining access permissions in the search definition.



**Note:** You may want to include the script administrator in an email notification for any time a saved search included in the script is updated. Email notifications can be defined on the Alerts tab of the saved search definition.

- On a Script record page, if you define a saved search as a List/Record script parameter, only saved searches that are public will appear in the List/Record parameter dropdown field. For information on script parameters, see [Creating Script Parameters Overview](#) in the NetSuite Help Center.

- In `nlapiSearchRecord(type, id, filters, columns)`, the value of `id` can be the ID that appears in the **Internal ID** column on the Saved Searches list page (see figure below). Or it can be the value that appears in the **ID** column.

If you have created a custom scriptId for your saved search, this will appear in the **ID** column (see figure). Note: To access the Saved Searches list page, go to Lists > Search > Saved Searches.

Saved Searches					
		FILTERS			
		USE	TYPE	ACCESS LEVEL	SCHEDULED
<input type="checkbox"/> SHOW ALL PRIVATE SEARCHES	General	- All -	- All -	- All -	- All -
<input type="checkbox"/> SHOW INACTIVES					
EDIT   RESULTS	NAME	FROM BUNDLE	ID	SEARCH FORM	TYPE
Edit   Results	New Corporate Leads	8259	customsearch18	Search Form	Customer
Edit   Results	AdvPromo Valid Item Types	49247	customsearch_advpromo_item_types	Search Form	Item
Edit   Results	Alternate Price List		customsearch102	Search Form	Item

In the following code, a Customer saved search is executed. The ID `customsearch57` references a specific saved search.

**Note:** The second parameter in `nlapiSearchRecord(...)` is treated as a variable instead of the custom Saved Search ID if it is not within quotes. Also note, if the Internal Id of the saved search is used instead of the Saved Search ID, the Internal Id does not need single quotes since it is evaluated as an integer.

```
function getEmail(firstname, lastname) {
    // Specify the record type and the saved search ID
    var searchresults = nlapiSearchRecord('customer', 'customsearch57', null, null);

    for ( var i = 0; searchresults != null && i < searchresults.length; i++ ) {
        var customerrecord = searchresults[i];

        if (customerrecord.getValue('firstname') == firstname && customerrecord.getValue('lastname') == lastname) {
            return customerrecord.getValue('email');
        }
    }
    return "Customer not found.";
}
```

## Filtering a Search

The following samples provide examples for how to set various kinds of filtering criteria in a search. Also provided are samples that show how to filter the results.

### Executing an Opportunity Search and Setting Search Filters

```
// 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' );

// Execute the search. You must specify the internal ID of the record type.
var searchresults = nlapiSearchRecord( 'opportunity', null, filters, columns );

// Loop through all search results. When the results are returned, use methods
// on the nlobjSearchResult object to get values for specific fields.
for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
{
    var searchresult = searchresults[ i ];
    var record = searchresult.getId( );
    var rectype = searchresult.getRecordType( );
    var salesrep = searchresult.getValue( 'salesrep' );
    var salesrep_display = searchresult.getText( 'salesrep' );
    var salesrep_email = searchresult.getValue( 'email', 'salesrep' );
    var customer = searchresult.getValue( 'entity' );
    var customer_email = searchresult.getValue( 'email', 'customer' );
    var expectedclose = searchresult.getValue( 'expectedclosedate' );
    var projectedamount = searchresult.getValue( 'projectedamount' );
    var probablity = searchresult.getValue( 'probability' );
}

```

## Executing an Opportunity Search and Setting 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 = 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');

```

```
//Execute the search. You must specify the internal ID of the record type.
var searchresults = nlapiSearchRecord('opportunity', null, filterExpression, columns);

//Loop through all search results. When the results are returned, use methods
//on the nlobjSearchResult object to get values for specific fields.
for (var i = 0; searchresults != null && i < searchresults.length; i++)
{
    var searchresult = searchresults[i];
    var record = searchresult.getId();
    var rectype = searchresult.getRecordType();
    var salesrep = searchresult.getValue('salesrep');
    var salesrep_display = searchresult.getText('salesrep');
    var salesrep_email = searchresult.getValue('email', 'salesrep');
    var customer = searchresult.getValue('entity');
    var customer_email = searchresult.getValue('email', 'customer');
    var expectedclose = searchresult.getValue('expectedclosedate');
    var projectedamount = searchresult.getValue('projectedamount');
    var probability = searchresult.getValue('probability');
}
```

## Filtering Based on Check Box Fields

When filtering search results for check box fields, use the **is** operator with **T** or **F** as the filter values. For example, in the following portlet script, all memorized Cash Sale transactions are returned.

```
function testPortlet(portlet) {
    portlet.setTitle('Memorized Cash Sales');

    var filters = new Array();
    filters[0] = new nlobjSearchFilter('name', null, 'equalTo', '87', null);
    filters[1] = new nlobjSearchFilter('memorized', null, 'is', 'T', null);

    var columns = new Array();
    columns[0] = new nlobjSearchColumn('internalid');
    columns[1] = new nlobjSearchColumn('memorized');

    var searchresults = nlapiSearchRecord('cashsale', null, filters, columns);
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
    {
        var searchResult = searchresults[i];
        portlet.addLine(i+": "+searchResult.getValue('internalid")+
        "+searchResult.getValue('memorized'),null,0);
    }
}
```

## Executing a Customer Search and Filtering the Results

In the following sample, a search for all customer records (leads, prospects, customers) in the system is executed with the maximum limit of 10 results set. Note that in this sample, if you specify customer as the record type, customers, leads, and prospects are returned in the results.

```
function executeSearch()
```



```
{
var searchresults = nlapiSearchRecord( 'customer', null, null, null );
for ( var i = 0; i < Math.min( 10, searchresults.length ); i++ )
{
    var record = nlapiLoadRecord(searchresults[i].getRecordType(), searchresults[i].getId());
}
}
```

## Filtering Based on None of Null Value

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@');
```

In the following example, only customer records that match the entityid of test1 are returned.

```
function filterCustomers()
{
    var filters = new Array();
    filters[0] = new nlobjSearchFilter( 'entityid', null, 'contains', 'test1', null );
    var searchresults = nlapiSearchRecord('customer', 11, filters, null);
    var emailAddress = "";
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
    {
        var searchresult = searchresults[ i ];
    }
}
```



**Note:** If it is unclear which values you can filter by for a specific filter variable, try performing a search that returns the value of a field so that you can see possible options.

## Returning Specific Fields in a Search

You can use the `nlobjSearchResult.getValue` method to return the values of specific record fields. In the following example, the email fields for records returned from an existing customer saved search are returned. The script ID of the saved search is customsearch8.

```
function findCustomerEmails()
{
    var searchresults = nlapiSearchRecord('customer', customsearch8, null,null);
    var emailAddress = "";
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
    {
        var searchresult = searchresults[ i ];
        emailAddress += searchresult.getValue( 'email' );
    }
}
```

For another example that shows how to return specific values in a search, see the sample for [Searching on Custom Records](#).

**Note:** To improve performance, if you only need to access a specific subset of fields, you should limit the returned objects to include only that subset. This can be accomplished using the `nlobjSearchFilter` and `nlobjSearchColumn(name, join, summary)` objects.

## Searching on Custom Records

Searching on custom records is the same as searching on standard (built-in) records. The following sample shows how to execute a search on a Warranty custom record type. (The internal ID for this record type is `customrecord_warranty`).

This sample shows how to define search filters and search columns, and then execute the search as you would for any other record type. This sample also shows how to use methods on the `nlobjSearchResult` object to get the values for the search results.

```
function searchWarranties()
{
    // define search filters
    var filters = new Array();
    filters[0] = new nlobjSearchFilter( 'created', null, 'onOrAfter', 'daysAgo15' );

    // return opportunity sales rep, customer custom field, and customer ID
    var columns = new Array();
    columns[0] = new nlobjSearchColumn( 'name' );
    columns[1] = new nlobjSearchColumn( 'owner' );
    columns[2] = new nlobjSearchColumn( 'custrecord_customer' );
    columns[3] = new nlobjSearchColumn( 'custrecord_resolutiontime' );

    // execute the Warrenty search, passing all filters and return columns
    var searchresults = nlapiSearchRecord( 'customrecord_warranty', null, filters, columns );

    // loop through the results
    for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
    {
        // get result values
        var searchresult = searchresults[ i ];
        var record = searchresult.getId();
        var rectype = searchresult.getRecordType();
        var name = searchresult.getValue( 'name' );
        var resolution = searchresult.getValue( 'custrecord_resolutiontime' );
        var customer = searchresult.getValue( 'custrecord_customer' );
        var customer_name = searchresult.getText( 'custrecord_customer' );
    }
}
```

This sample shows how you can do a search on custom records using a search filter expression.

```
function searchWarranties()
{
```



```

//Define search filter expression
var filterExpression = ['created', 'onOrAfter', 'daysAgo15'];

//Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn('name');
columns[1] = new nlobjSearchColumn('owner');
columns[2] = new nlobjSearchColumn('custrecord_customer');
columns[3] = new nlobjSearchColumn('custrecord_resolutiontime');

//Execute the Warranty search, passing search filter expression and columns
var searchresults = nlapiSearchRecord('customrecord_warranty', null, filterExpression, columns);

//Loop through the results
for (var i = 0; searchresults != null && i < searchresults.length; i++)
{
    //Get result values
    var searchresult = searchresults[i];
    var record = searchresult.getId();
    var rectype = searchresult.getType();
    var name = searchresult.getValue('name');
    var resolution = searchresult.getValue('custrecord_resolutiontime');
    var customer = searchresult.getValue('custrecord_customer');
    var customer_name = searchresult.getText('custrecord_customer');
}
}

```

## Searching Custom Lists

The following sample shows how to search a custom list.

```

var col = new Array();
col[0] = new nlobjSearchColumn('name');
col[1] = new nlobjSearchColumn('internalId');
var results = nlapiSearchRecord('customlist25', null, null, col);
for (var i = 0; results != null && i < results.length; i++)
{
    var res = results[i];
    var listValue = (res.getValue('name'));
    var listID = (res.getValue('internalId'));
    nlapiLogExecution('DEBUG', (listValue + ", " + listID));
}

```

## Executing Joined Searches

This example shows how to set values for a joined search. In this case you are executing an Item search that uses **Customer** and **Currency** (as specified on the Pricing record) as your filtering criteria.

You will define the join to the Pricing record in the `nlobjSearchFilter` object. You will define search return column values (also joins to the Pricing record) in the `nlobjSearchColumn(name, join, summary)` object. You will execute the **Item** search using `nlapiSearchRecord(type, id, filters, columns)`.

```
// Create a filters array and define search filters for an Item search
var filters = new Array();

// filter by a specific customer (121) on the Pricing record
filters[0] = new nlobjSearchFilter('customer', 'pricing', 'is', '121');

// filter by a currency type (USA) on the Pricing record
filters[1] = new nlobjSearchFilter('currency', 'pricing', 'is', '1');

// set search return columns for Pricing search
var columns = new Array();

// return data from pricelevel and unitprice fields on the Pricing record
columns[0] = new nlobjSearchColumn('pricelevel', 'pricing');
columns[1] = new nlobjSearchColumn('unitprice', 'pricing');

// specify name as a search return column. There is no join set in this field.
// This is the Name field as it appears on Item records.
columns[2] = new nlobjSearchColumn('name');

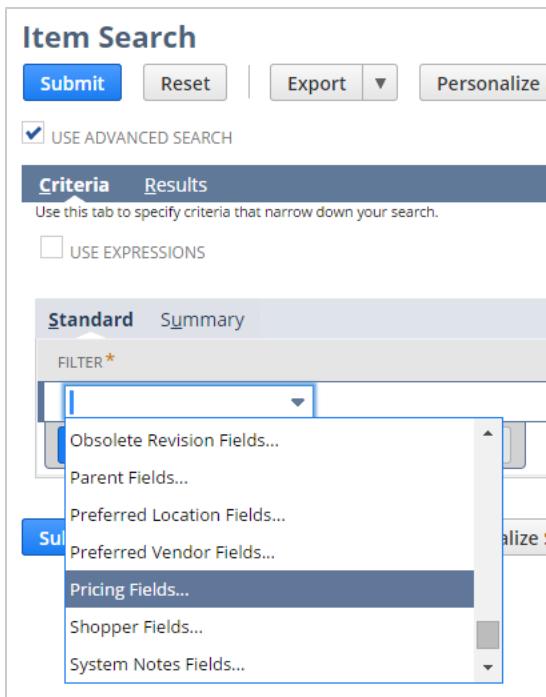
// execute the Item search, which uses data on the Pricing record as search filters
var searchresults = nlapiSearchRecord('item', null, filters, columns);
```

The following figures show the UI equivalent of executing an Item search that uses filtering criteria pulled from the Pricing record. Note that on the Criteria tab, all available search joins for an Item search will appear at the bottom of the Filter drop-down list. Available join records are marked with the ellipsis (...) after the record name.

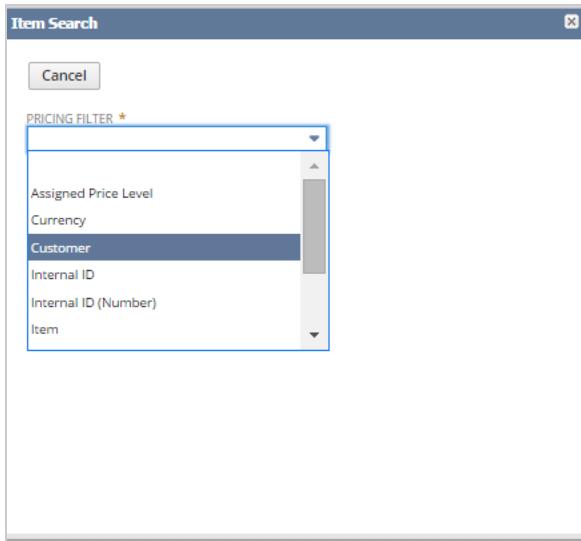
**Note:** Not all joins that appear in the UI are supported in SuiteScript. To see which joins are supported for a particular search, start by going to [SuiteScript Supported Records](#). Click the record type that you want to execute the search on. Based on the example described below, you will click **Item Search** record. Then look to see which joins are supported for the record type.

The figures below show only how to set the filter values for a joined search. All of the same concepts apply when specifying search return column values.

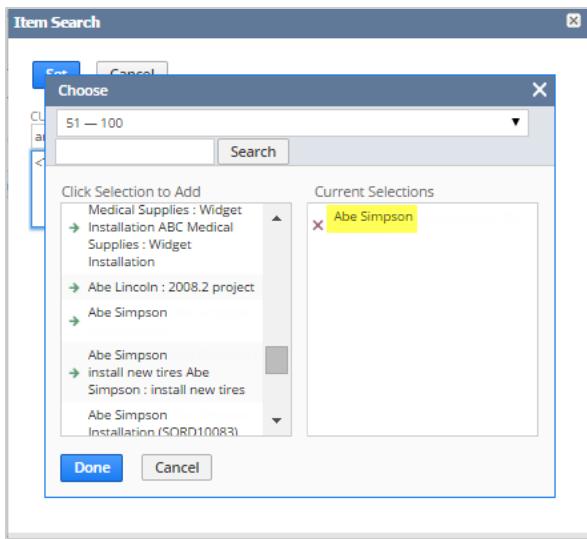
The first figure shows the Item Search record (Lists > Accounting > Items > Search).



When **Pricing Fields...** is selected, a popup appears with all search fields that are available on the Pricing record (see figure below).



When you select the **Customer** field, another popup opens allowing you to select one or more customers. In the code sample, customer 121 is specified. In the UI, this customer appears as Abe Simpson (see below).



This figure shows how Item search / pricing join filtering criteria appear in the UI.

The screenshot shows the 'Criteria' tab of a search interface. It includes a checkbox for 'USE EXPRESSIONS' and two rows of search filters:

FILTER *	DESCRIPTION *
Pricing : Customer	is Abe Simpson
Pricing : Currency	is USA

In SuiteScript, this looks like:

```
var filters = new Array();
filters[0] = new nlobjSearchFilter('customer', 'pricing', 'is', '121');
filters[1] = new nlobjSearchFilter('currency', 'pricing', 'is', '1');
```

This example shows how you can execute joined searches using a search filter expression.

```
//Define search filter expression
var filterExpression =      [ [ 'pricing.customer', 'is', 121 ],
                            'and',
                            [ 'pricing.currency', 'is', 1 ] ];

//Define search columns
var columns = new Array();
//Return data from pricelevel and unitprice fields on the Pricing record
columns[0] = new nlobjSearchColumn('pricelevel', 'pricing');
columns[1] = new nlobjSearchColumn('unitprice', 'pricing');
//Specify name as a search return column. There is no join set in this field.
//This is the Name field as it appears on Item records.
columns[2] = new nlobjSearchColumn('name');

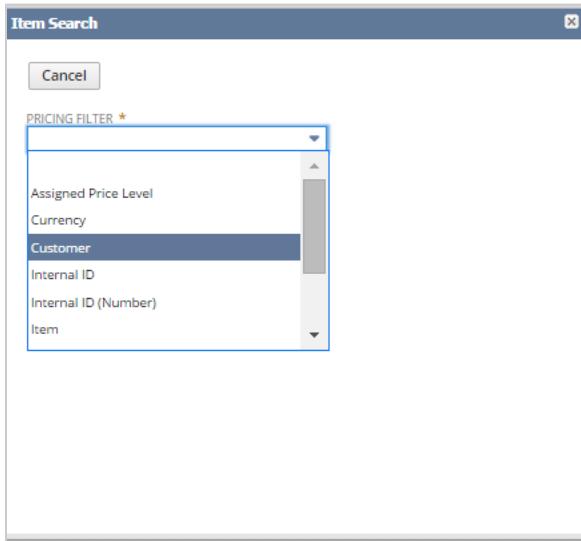
//Execute the Item search, which uses data on the Pricing record as search filter expression
var searchresults = nlapiSearchRecord('item', null , filterExpression, columns);
```

The following figures show the UI equivalent of executing the preceding example. These figures show only how to set the filter expression for a joined search. All of the same concepts apply when specifying search return column values.

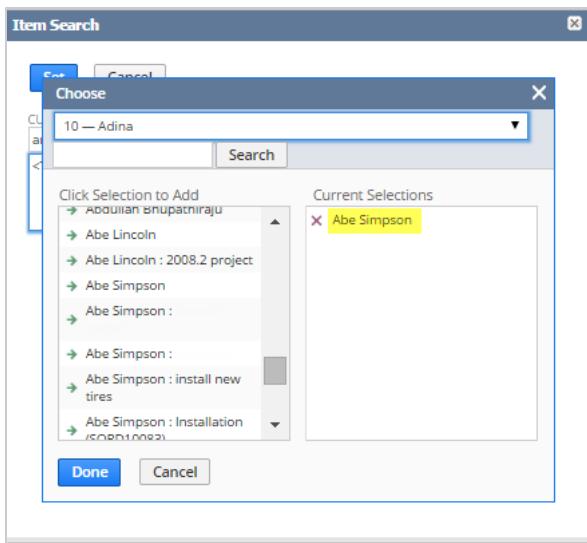
The first figure shows the Item Search record (Lists > Accounting > Items > Search).

The screenshot shows the 'Item Search' record interface. At the top, there are buttons for 'Submit', 'Reset', 'Export', 'Personalize Search', and 'Create Saved Search'. Below these are two checkboxes: 'USE ADVANCED SEARCH' and 'USE EXPRESSIONS'. The main area is titled 'Criteria' and contains tabs for 'Standard' and 'Summary'. A dropdown menu is open over the 'Pricing Fields...' option in the 'FILTER \*' column. The menu includes options like 'Member Item Fields...', 'Obsolete Revision Fields...', 'Parent Fields...', 'Preferred Location Fields...', 'Preferred Vendor Fields...', 'Pricing Fields...', and 'Shopper Fields...'. At the bottom of the interface are 'Submit' and 'Reset' buttons.

When **Pricing Fields...** is selected, a popup appears with all search fields that are available on the Pricing record. (See figure below.)



When you select the Customer field, another popup opens allowing you to select one or more customers. In the code sample, customer 121 is specified. In the UI, this customer appears as Abe Simpson. (See figure below.)



The following figure shows how Item search / pricing join filtering criteria appear in the UI.

The screenshot shows the 'Criteria' tab of a search interface. It includes a note: 'Use this tab to specify criteria that narrow down your search.' A checked checkbox labeled 'USE EXPRESSIONS' is present. Below is a table:

Standard	Summary		
NOT	PARENTS	FILTER *	DESCRIPTION *
	((	Pricing : Customer	is Abe Simpson
	(	Pricing : Currency	is US Dollar

## Searching for an Item ID

The following search sample returns an array of item search results matched by the "Item ID" keyword. You can then iterate through each result and call `nlobjSearchResult.getId()` to get the internal item ID.

```
var x = nlapiSearchRecord('item', null, new nlobjSearchFilter('itemid', null, 'haskeywords', it
emidvalue));
```

## Searching for Duplicate Records

In the course of doing business, it is common to have more than one record created for the same contact, customer, vendor, or partner. In both the UI and in SuiteScript you can find all duplicate records after your NetSuite administrator has enabled the Duplicate Detection & Merge feature (Setup > Company > Enable Features, on the Company subtab, Data Management section).

In SuiteScript, a search for duplicate records is executed using the `nlapiSearchDuplicate(type, fields, id)` function. For the definition of this API, as well as a code sample, see [nlapiSearchDuplicate\(type, fields, id\)](#).



**Note:** For general information on searching for duplicate records in NetSuite, see the help topic [Duplicate Record Detection](#) in the NetSuite Help Center.

## Performing Global Searches

With NetSuite's global search, you can find records from anywhere in your account data. In the UI, you enter your search keywords in the **Search** field in the upper right corner of any page.

In SuiteScript, global searches are executed using the `nlapiSearchGlobal(keywords)` function. For the definition of this API, as well as a code sample, see [nlapiSearchGlobal\(keywords\)](#).



**Note:** For general information on global searching in NetSuite, see the help topic [Global Search](#) in the NetSuite Help Center.

## Searching CSV Saved Imports

You can use SuiteScript to search the CSV saved imports in an account. Executing this kind of search may be useful if you need to access import information for CSV saved imports that were bundled and installed in a new account.

For the **type** argument in `nlapiSearchRecord(type, id, filters, columns)`, you will set `savedcsvimport`. For example:

```
var search = nlapiSearchRecord('savedcsvimport', null, null, null);
```

Note that `savedcsvimport` is not a true record type in NetSuite, as it cannot be manipulated in SuiteScript or in web services. The `savedcsvimport` type can be used only in search. The available filter and return values for `savedcsvimport` are:

- `internalid`
- `name`
- `description`

## Using Formulas, Special Functions, and Sorting in Search

```
function searchRecords()
{
    //specify a formula column that displays the name as: Last Name, First Name (Middle Name)
    var name = new nlobjSearchColumn('formulatext');
    name.setFormula("'{lastname}||', '{firstname}||case when LENGTH({middlename})=0 then '' else '||{middlename}||' end");

    //now specify a numeric formula field
    var number = new nlobjSearchColumn('formulanumeric').setFormula("1234.5678");

    //now specify a numeric formula field and format the output using a special function
}
```

```

var roundednumber = new nlobjSearchColumn('formulanumeric').setFormula("1234.5678").setFunction("round");

//now specify a sort column (sort by internal ID ascending)
var internalid = new nlobjSearchColumn('internalid').setSort(false /* bsortdescending */);

var columns = [name, number, roundednumber, internalid];
var filterHasMiddleName = new nlobjSearchFilter('middlename', null, 'isNotEmpty');

var searchresults = nlapiSearchRecord('contact', null, filterHasMiddleName, columns);
for (var i = 0; i < searchresults.length; i++)
{
    //access the value using the column objects
    var contactname = searchresults[i].getValue(name);
    var value = searchresults[i].getValue(number);
    var valuerounded = searchresults[i].getValue(roundednumber);
}
}

```

## Using Summary Filters in Search

```

function searchRecords()
{
    //perform a summary search: return all sales orders total amounts by customer for those with
    total sales > 1000
    var filter = new nlobjSearchFilter('amount', null, 'greaterThan', 1000).setSummaryType('sum'
    );
    var entity = new nlobjSearchColumn('entity', null, 'group');
    var amount = new nlobjSearchColumn('amount', null, 'sum');

    var searchresults = nlapiSearchRecord('salesorder', null, filter, [entity, amount]);
    for (var i = 0; i < searchresults.length; i++)
    {
        //access the values this time using the name and summary type
        var entity = searchresults[i].getValue('entity', null, 'group');
        var entityName = searchresults[i].getText('entity', null, 'group');
        var amount = searchresults[i].getValue('amount', null, 'sum');
    }
}

```

# Supported Search Operators, Summary Types, and Date Filters



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section lists all NetSuite field types that support SuiteScript search, as well as the operators that can be used on each type. Also listed are supported search summary types and search date filters.

See the following sections for detailed reference information:

- [Search Operators](#)
- [Search Summary Types](#)
- [Search Date Filters](#)

## Search Operators



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists each field type and its supported search operator.

Search Operator	List/Record	Currency, Decimal Number, Time of Day	Date	Check Box	Document, Image	Email Address, Free-Form Text, Long Text, Password, Percent, Phone Number, Rich Text, Text Area,	Multi Select
after			X				
allof						X	
any		X			X		
anyof	X				X		X
before			X				
between		X					
contains					X		
doesnotcontain					X		
doesnotstartswith					X		
equalto		X					
greaterthan		X					
greaterthanorequalto		X					
haskeywords					X		
is				X		X	
isempty		X	X			X	
isnot						X	
isnotempty		X	X			X	
lessthan		X					
lessthanorequalto		X					
noneof	X				X		X
notafter			X				
notallof							X
notbefore			X				
notbetween		X					

Search Operator	List/Record	Currency, Decimal Number, Time of Day	Date	Check Box	Document, Image	Email Address, Free-Form Text, Long Text, Password, Percent, Phone Number, Rich Text, Text Area,	Multi Select
notequalto		X					
notgreaterthan		X					
notgreaterthanorequalto		X					
notlessthan		X					
notlessthanorequalto		X					
notin			X				
notinonrafter			X				
notinonrbefore			X				
notinwithin			X				
on			X				
onorafter			X				
onorbefore			X				
startswith						X	
within			X				

## Search Summary Types

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists the summary types that can be applied to organize your search results. Note that these summary types are available on the Results tab in the UI.

Summary Type	Type Internal ID	Purpose	Example
Group	group	Rolls up search results under the column to which you apply this type.	In a search for sales transactions, you can group the transactions found by customer name.
Count	count	Counts the number of results found that apply to this column.	In a search of items purchased by customers, you can view a count of the number of items purchased by each customer.
Sum	sum	Sums search results.	In a search of purchases this period, you can total the Amount column on your results.
Minimum	min	Shows the minimum amount found in search results.	In a search of sales transactions by sales rep, you can show the minimum amount sold in the transaction.
Maximum	max	Shows the maximum amount found in search results.	In a customer search by partner, you can show the maximum amount of sales by each partner.
Average	avg	Calculates the average amount found in your search results.	In an employee search, you can average the amounts of

Summary Type	Type Internal ID	Purpose	Example
			your employees' company contributions.

## Search Date Filters

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists all supported search date filters. Values are not case sensitive

**i Note:** These values correspond to selectors used in SuiteAnalytics. For additional information on these values, see the help topics [Period Selectors](#), [Date Range Selectors](#), and [Date As Of Selectors](#).

fiscalHalfBeforeLast	nextBusinessWeek	sameFiscalHalfLastFiscalYear	startOfThisBusinessWeek
fiscalHalfBeforeLastToDate	nextFiscalHalf	sameFiscalHalfLastFiscalYearToDate	startOfThisFiscalHalf
fiscalQuarterBeforeLast	nextFiscalQuarter	sameFiscalQuarterLastFiscalYear	startOfThisFiscalQuarter
fiscalQuarterBeforeLastToDate	nextFiscalYear	sameFiscalQuarterFiscalYearBeforeLast	startOfThisFiscalYear
fiscalYearBeforeLast	nextFourWeeks	sameFiscalQuarterFiscalYearBeforeLast	startOfThisMonth
fiscalYearBeforeLastToDate	nextMonth	sameFiscalQuarterLastFiscalYear	startOfThisWeek
fiveDaysAgo	nextOneHalf	sameFiscalQuarterLastFiscalYearToDate	startOfThisYear
fiveDaysFromNow	nextOneMonth	sameMonthFiscalQuarterBeforeLast	startOfWeekBeforeLast
fourDaysAgo	nextOneQuarter	sameMonthFiscalYearBeforeLast	tenDaysAgo
fourDaysFromNow	nextOneWeek	sameMonthLastFiscalQuarterToDate	tenDaysFromNow
fourWeeksStartingThisWeek	ninetyDaysAgo	sameMonthLastFiscalYearBeforeLast	thirtyDaysAgo
lastBusinessWeek	ninetyDaysFromNow	sameMonthLastFiscalYearToDate	thirtyDaysFromNow
lastFiscalHalf	oneYearBeforeLast	sameMonthLastFiscalYearToLast	thisBusinessWeek
lastFiscalHalfOneFiscalYearAgo	previousFiscalQuartersLastFiscalYear	sameMonthLastFiscalYearToYear	thisFiscalHalf
lastFiscalHalfToDate	previousFiscalQuartersThisFiscalYear	sameMonthLastFiscalYearToYear	thisFiscalHalfToDate
lastFiscalQuarter	previousMonthsLastFiscalHalf	sameMonthLastFiscalYearYear	thisFiscalQuarter
lastFiscalQuarterOneFiscalYearAgo	previousMonthsLastFiscalQuarter	sameMonthLastFiscalYearYear	thisFiscalQuarterToDate
lastFiscalQuarterToDate	previousMonthsLastFiscalYear	sameWeekFiscalYearBeforeLast	thisFiscalYear
lastFiscalQuarterTwoFiscalYearsAgo	previousMonthsLastFiscalYear	sameWeekLastFiscalYear	thisFiscalYearToDate
lastFiscalYear	previousMonthsSameFiscalHalfLastFiscalYear	sixtyDaysAgo	thisMonth
lastFiscalYearToDate	previousMonthsSameFiscalQuarterLastFiscalYear	sixtyDaysFromNow	thisMonthToDate
lastMonth	previousMonthsThisFiscalHalf	startOfFiscalHalfBeforeLast	thisRollingHalf
lastMonthOneFiscalQuarterAgo	previousMonthsThisFiscalQuarter	startOfFiscalQuarterBeforeLast	thisRollingQuarter
lastMonthToDate	previousMonthsThisFiscalYear	startOfFiscalYearBeforeLast	thisRollingYear
lastMonthTwoFiscalQuartersAgo	previousOneDay	startOfLastBusinessWeek	thisWeek
lastMonthTwoFiscalYearsAgo	previousOneHalf	startOfLastFiscalHalf	thisWeekToDate
lastRollingHalf	previousOneMonth	startOfLastFiscalHalfOneFiscalYearAgo	thisYear
lastRollingQuarter	previousOneQuarter	startOfLastFiscalQuarter	threeDaysAgo
lastRollingYear	previousOneWeek	startOfLastFiscalQuarterOneFiscalYearAgo	threeDaysFromNow
lastWeek		startOfLastFiscalYear	threeFiscalQuartersAgo

lastWeekToDate	previousOneYear	startOfLastMonth	twoDaysFromNow
monthAfterNext	previousRollingHalf	startOfLastMonthOneFis	weekAfterNext
monthAfterNextToDate	previousRollingQuarter	calQuarterAgo	weekAfterNextToDate
monthBeforeLast	previousRollingYear	startOfLastMonthOneFis	weekBeforeLast
monthBeforeLastToDate	sameDayFiscalQuarterBeforLast	calYearAgo	weekBeforeLastToDate
	sameDayFiscalYearBeforeLast	startOfLastRollingHalf	yesterday
	sameDayLastFiscalQuarter	startOfLastRollingQuarter	
	sameDayLastFiscalYear	startOfLastRollingYear	
	sameDayLastMonth	startOfLastWeek	
	sameDayLastWeek	startOfMonthBeforeLast	
	sameDayMonthBeforeLast	startOfNextBusinessWeek	
	sameDayWeekBeforeLast	startOfNextFiscalHalf	
		startOfNextFiscalQuarter	
		startOfNextFiscalYear	
		startOfNextMonth	
		startOfNextWeek	
		startOfPreviousRollingHalf	
		startOfPreviousRollingQuarter	
		startOfPreviousRollingYear	
		startOfSameFiscalHalfLastFiscalYear	
		startOfSameFiscalQuarterLastFiscalYear	
		startOfSameMonthLastFiscalQuarter	
		startOfSameMonthLastFiscalYear	

## Deprecated Search Date Filters



**Important:** The following search date filters are deprecated as of Version 2015 Release 1. These values are still supported in existing scripts. For new scripts, please use the arguments listed in the above table.

lastFiscalHalfOneYearAgo	startOfSameHalfLastFiscalYear
lastFiscalQuarterOneYearAgo	startOfSameQuarterLastFiscalYear
lastMonthOneQuarterAgo	startOfTheHalfBeforeLast
lastMonthOneYearAgo	startOfTheMonthBeforeLast
lastMonthTwoQuartersAgo	startOfTheQuarterBeforeLast
lastMonthTwoYearsAgo	startOfTheWeekBeforeLast
lastQuarterTwoYearsAgo	startOfTheYearBeforeLast
previousMonthsSameFiscalHalfLastYear	threeQuartersAgo
previousMonthsSameFiscalQuarterLastFiscalYear	threeQuartersAgoToDate
previousQuartersLastFiscalYear	threeYearsAgo
previousQuartersThisFiscalYear	threeYearsAgoToDate
sameHalfLastFiscalYear	daysagoxx
sameHalfLastFiscalYearToDate	weeksagoxx
sameQuarterLastFiscalYear	monthsagoxx
sameQuarterLastFiscalYearToDate	quartersagoxx
startOfLastHalfOneYearAgo	yearsagoxx
startOfLastMonthOneQuarterAgo	daysfromnowxx

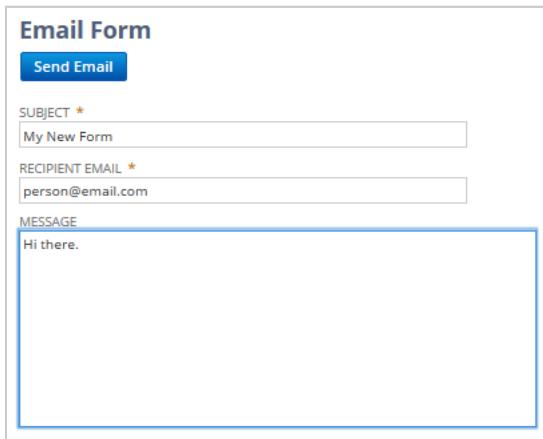
startOfLastMonthOneYearAgo startOfLastQuarterOneYearAgo startOfNextYear	weeksfromnowxx monthsfromnowxx quartersfromnowxx yearsfromnowxx
---	--

# UI Objects Overview

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](#). SuiteScript UI objects are generated on the server as HTML. They are then displayed in the browser and are accessible through client scripts.

 **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.

The following figure is an email form Suitelet built with UI objects. The form itself is represented by the [nlobjForm](#) UI object. The Subject, Recipient email, and Message fields are represented by the [nlobjField](#) UI object, and the Send Email button is represented by the [nlobjButton](#) UI object.



The screenshot shows a Suitelet titled "Email Form". It contains a "Send Email" button at the top. Below it are three input fields: "SUBJECT \*", which has "My New Form" typed into it; "RECIPIENT EMAIL \*", which has "person@email.com" typed into it; and "MESSAGE", which has "Hi there." typed into it. All fields have a blue border.

To learn more about working with UI objects, see these topics:

- [Creating Custom NetSuite Pages with UI Objects](#)
- [InlineHTML UI Objects](#)
- [Building a NetSuite Assistant with UI Objects](#)

## Email Form Code

```
/**  
 * Build an email form Suitelet with UI objects. The Suitelet sends out an email  
 * from the current user to the recipient email address specified on the form.  
 */  
function simpleEmailForm(request, response)  
{  
    if ( request.getMethod() == 'GET' )  
    {  
        var form = nlapiCreateForm('Email Form');  
        var subject = form.addField('subject','text', 'Subject');  
        subject.setLayoutType('normal','startcol');  
        subject.setMandatory( true );  
        var recipient = form.addField('recipient','email', 'Recipient email');
```

```
recipient.setMandatory( true );
var message = form.addField('message','textarea', 'Message');
message.setDisplaySize( 60, 10 );
form.addSubmitButton("Send Email");

response.writePage(form);
}

else
{
    var currentuser = nlapi GetUser();
    var subject = request.getParameter('subject')
    var recipient = request.getParameter('recipient')
    var message = request.getParameter("message")
    nlapiSendEmail(currentuser, recipient, subject, message);
}
}
```

# Creating Custom NetSuite Pages with UI Objects

SuiteScript [UI Objects](#) encapsulate the UI elements necessary for building NetSuite-looking portlets, forms, fields, sublists, tabs, lists, columns, and assistant. Note that when developing a Suitelet with UI objects, you can also add custom fields with inline HTML.

Depending on the design and purpose of the custom UI, you can use either the [nlobjForm](#) UI object or [nlobjList](#) UI object as the basis. These objects encapsulate a scriptable NetSuite form and NetSuite list, respectively. You can then add a variety of scriptable UI elements to these objects to adopt the NetSuite look-and-feel. These elements can include fields (through [nlobjField](#)), buttons (through [nlobjButton](#)), tabs (through [nlobjTab](#)), and sublists (through [nlobjSubList](#)).



**Important:** When adding UI elements to an [existing page](#), you must prefix the element name with `custpage`. This minimizes the occurrence of field/object name conflicts. For example, when adding a custom tab to a NetSuite entry form in a user event script, the name should follow a convention similar to `custpage customtab` or `custpage mytab`.

## UI Objects and Suitelets

In Suitelet development, UI objects allow you to programmatically build custom NetSuite-looking pages. A blank [nlobjForm](#) object is created with [nlapiCreateForm\(title, hideNavbar\)](#). If you are building a custom assistant, a blank [nlobjAssistant](#) object is created with [nlapiCreateAssistant\(title, hideHeader\)](#). On the server, the Suitelet code adds fields, steps, tabs, buttons and sublists to the form and assistant objects.

The server defines the client script (if applicable) and sends the page to the browser. When the page is submitted, the values in these UI objects become part of the request and available to aid logic branching in the code.

For a basic example of a Suitelet built entirely of UI objects, see [What Are Suitelets?](#). This section also provides the code used to create the Suitelet. To see examples of an assistant or “wizard” Suitelet built with UI objects, see [Using UI Objects to Build an Assistant](#).

## UI Objects and User Event Scripts

On entry forms and transaction forms, the [nlobjForm](#) object is accessed in user events scripts on which new fields are added on the server before the pages are sent to the browser. With the NetSuite [nlobjForm](#) object exposed, you can design user event scripts that manipulate most built-in NetSuite UI components (for example, fields, tabs, sublists).



**Note:** If you are not familiar with concept of NetSuite entry or transaction forms, see the help topic [Custom Forms](#) in the NetSuite Help Center.

The key to using user event scripts to customize a form during runtime is a second argument named `form` in the **before load** event. This optional argument is the reference to the entry/transaction form. You can use this to dynamically change existing form elements, or add new ones (see [Enhancing NetSuite Forms with User Event Scripts](#)).



**Note:** Sometimes the best solution for a customized workflow with multiple pages is a hybrid UI design, which encompasses both customized entry forms as well as Suitelets built with UI objects.

## UI Object Descriptions and Examples

For details about the UI objects supported by NetSuite, as well as code samples, see [UI Objects](#).



# InlineHTML UI Objects

SuiteScript UI objects make most of the NetSuite UI elements scriptable. However, they still may not lend themselves well to certain use cases. In these circumstances, developers can create custom UI elements by providing HTML that SuiteScript can render on a NetSuite page. These UI elements are known as “InlineHTML”.

InlineHTML can be implemented in two ways:

1. Pure custom HTML with no SuiteScript UI objects
2. Hybrid of custom HTML and SuiteScript UI objects



The first approach, shown on the left side, requires you to provide all the HTML code you want to appear on the page, as if performing web designing on a blank canvas. The second approach, shown on the right, allows custom HTML to be embedded in a NetSuite page. Example code is available in the help section for [Suitelets Samples](#).

A blog hosted within NetSuite can be created with a hybrid of inlineHTML and UI objects. A blog page can display blog entries in descending chronological order with “Read More” hyperlinks. Pagination should be used to handle the potentially large number of entries. Readers should also be able to read and leave comments. These requirements cannot be satisfied by standard NetSuite UI elements in a reader-friendly manner. However, rendering the blog entries’ data (stored as custom records) in HTML and displaying it in SuiteScript UI objects as inlineHTML would satisfy this use case.

# Building a NetSuite Assistant with UI Objects

- [NetSuite UI Object Assistant Overview](#)
- [Understanding NetSuite Assistants](#)
- [Using UI Objects to Build an Assistant](#)

## NetSuite UI Object Assistant Overview

You can use UI objects to build an assistant or “wizard” within NetSuite that has the same look-and-feel as other built-in NetSuite assistants. To build your own assistant, you will use objects such as [nlobjAssistant](#), [nlobjAssistantStep](#), [nlobjFieldGroup](#), and [nlobjField](#).

For examples that show some of the built-in assistants that come with NetSuite, see [Understanding NetSuite Assistants](#).

To learn to how programmatically construct your own assistant, see [Using UI Objects to Build an Assistant](#).

## Understanding NetSuite Assistants

In NetSuite, assistants contain a series of steps that users must complete to accomplish a larger task. In some assistants, users must complete the steps sequentially. In others, steps are non-sequential, and they do not all have to be completed. In these assistants, steps are provided only as guidelines for actions users might want to take to complete a larger task.

The UI objects you use to construct your own assistant will encapsulate the look-and-feel of assistants already built in to NetSuite. For examples of these assistants, see these topics:

- [Web Site Assistant](#)
- [SuiteBundler Assistant](#)
- [Import Assistant](#)

## Web Site Assistant

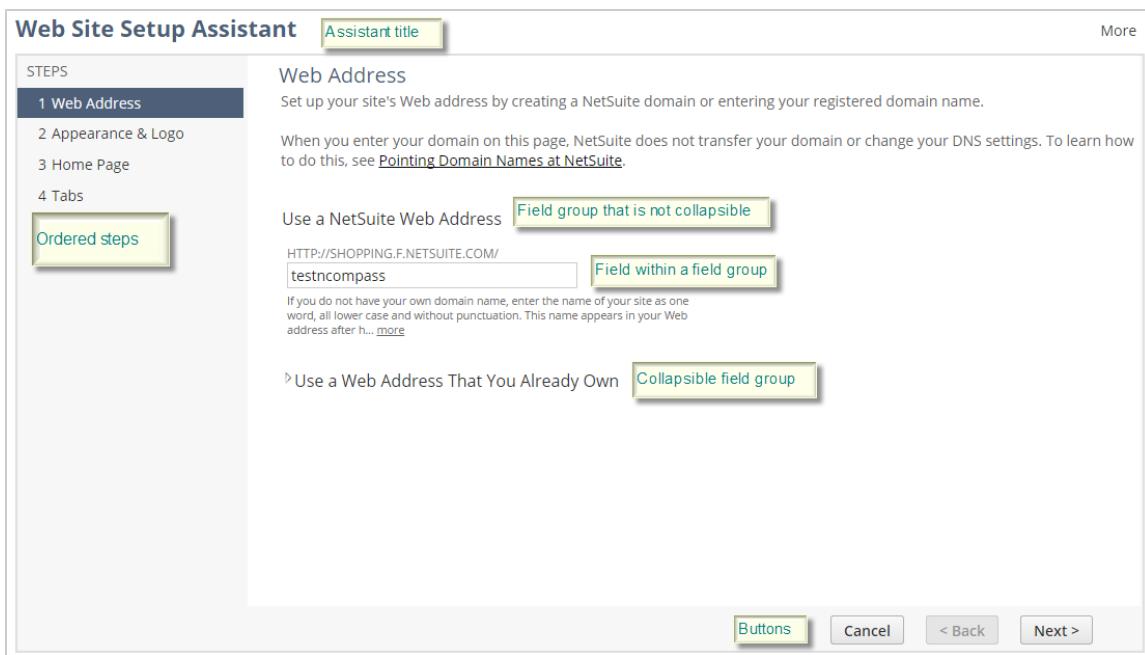
The Web Site Assistant is a built-in NetSuite assistant. This assistant guides users through a set steps that are ordered sequentially. The ultimate goal of the assistant is to help users build their own web sites.



**Note:** To access the Web Site Assistant, go to Setup > Site Builder > Web Site Assistant.

This figure shows Step 1 (page 1) of the Web Site Assistant. Steps are ordered sequentially and positioned vertically in the left panel. The current step is highlighted in gray.

All components called out in this figure can be built in a custom assistant.



## SuiteBundler Assistant

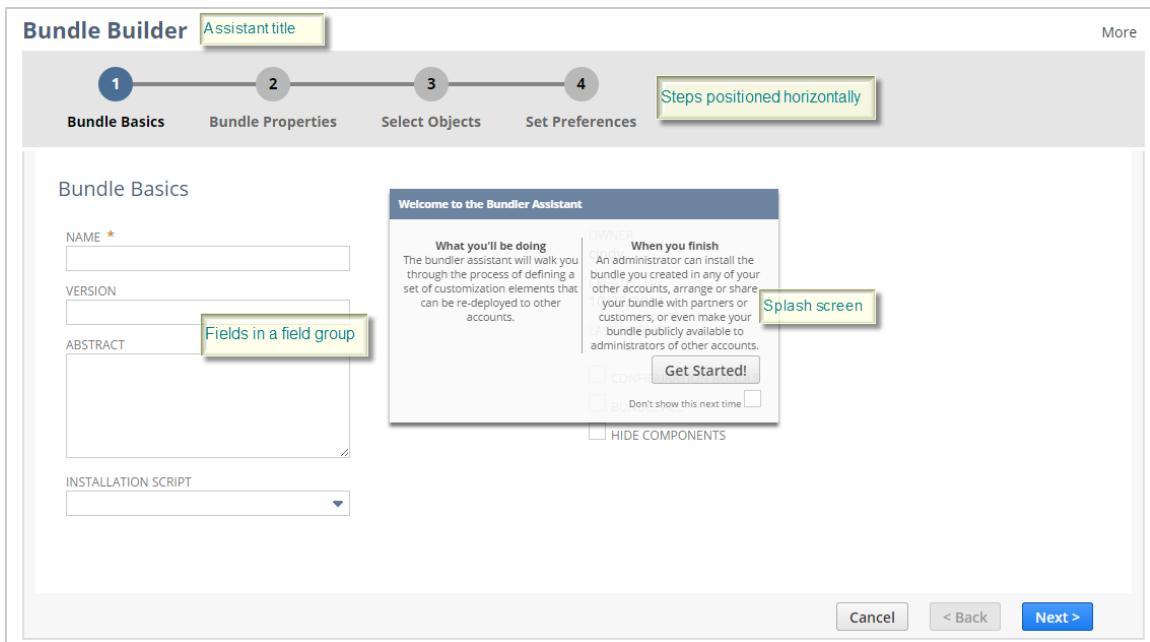
The SuiteBundler Assistant is another built-in NetSuite assistant. This assistant guides users through a set of steps in custom NetSuite solutions are “bundled,” later to be deployed into other NetSuite accounts.



**Note:** To access the SuiteBundler Assistant, go to Customization > SuiteBundler > Create Bundle.

This figure shows Step 1 (page 1) of the SuiteBundler Assistant. Steps are ordered sequentially and appear horizontally, directly below the title of the assistant.

All components called out in this figure can be built in a custom assistant.

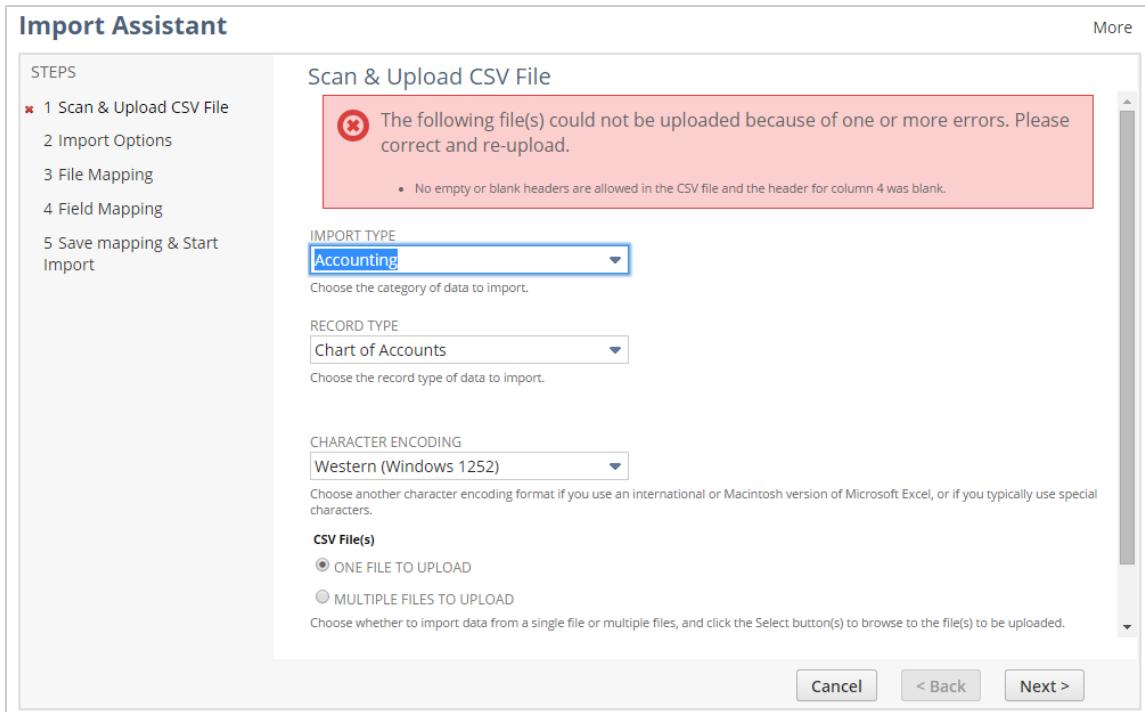


## Import Assistant

The Import Assistant guides users through a set steps that allow them to import data into NetSuite.

**Note:** To access the Import Assistant, go to Setup > Import/Export > Import CSV Records.

This figure shows what an error message looks like in an assistant. Users cannot proceed to the next step until the error is resolved. When building custom assistants, you can also throw errors that prevent users from moving to the next step.



## Using UI Objects to Build an Assistant

From a UI perspective, the building blocks of most assistants you build are going to include a combination of the following: [Steps](#), [Field Groups](#), [Fields](#), [Buttons](#), [Sublists](#).

To create this look, you will use objects such as [nlobjAssistant](#), [nlobjAssistantStep](#), [nlobjFieldGroup](#), and [nlobjField](#). The API documentation for each object and all object methods provides examples for building instances of each objects.

Also see these topics for additional information:

- [Understanding the Assistant Workflow](#)
- [Using Redirection in an Assistant Workflow](#)
- [Assistant Components and Concepts](#)
- [UI Object Assistant Code Sample](#)

Note that since your assistant is a [Suitelet](#), after you have finished building the assistant, you can initiate the assistant by creating a custom menu link that contains the Suitelet URL. (See [Running a Suitelet in NetSuite](#) for details on creating custom menu links for Suitelets.)

## Understanding the Assistant Workflow

If you have not done so already, please see [Using UI Objects to Build an Assistant](#) for information on the UI objects that are used to create an assistant.

In your SuiteScript code, there is an order in which many components of an assistant must be added. At a minimum you will:

1. [Create a new assistant](#)

- a. Call `nlapicreateassistant(title, hideHeader)`.
  - b. Add steps to the assistant. Use `nlobjassistant.addStep(name, label)`.
  - c. Define whether the steps must be completed sequentially or whether they can be completed in random order. Use `nlobjassistant.setOrdered(ordered)`.
2. **Build assistant pages**

Add fields, field groups, and sublists to build assistant pages for each step.



**Note:** In the context of an assistant, each step is considered a page.

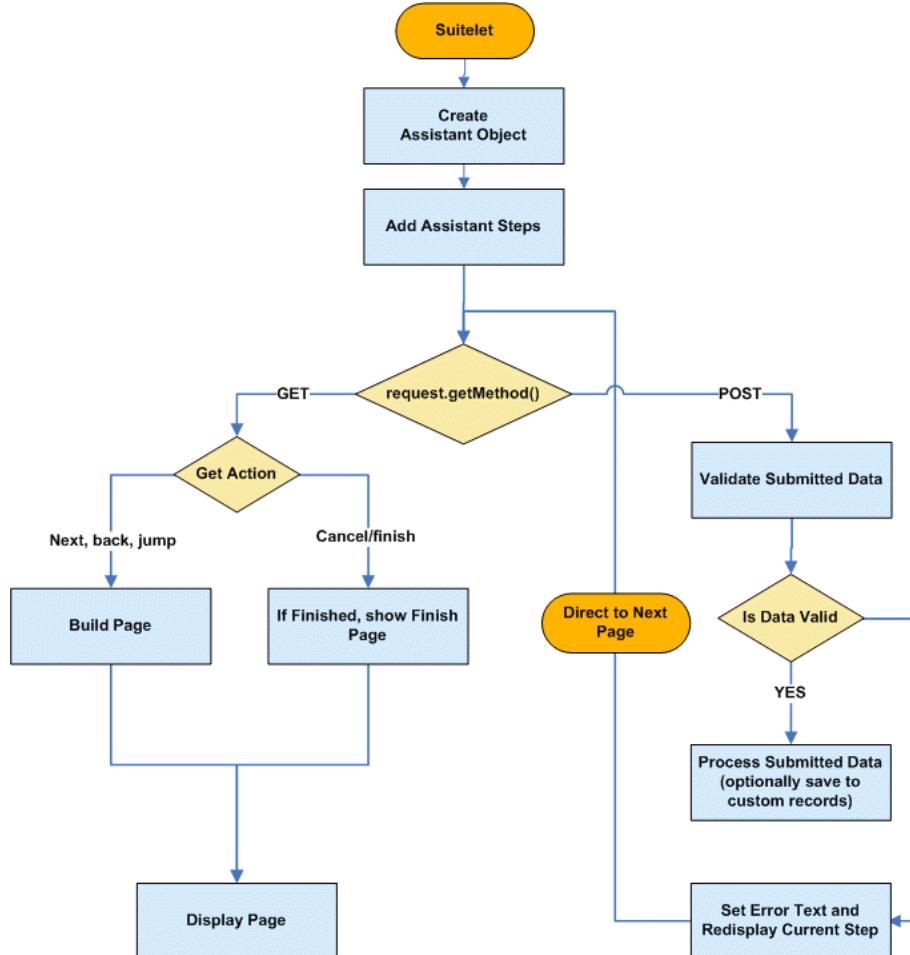
In the assistant workflow diagram (below), see **Build Page** for a list of methods that can be used to build a page.

### 3. Process assistant pages

In your Suitelet, construct pages in response to a user's navigation of the assistant. At a minimum you will render a specific assistant step/page using a GET request. Then you will process that page in the POST request, before then redirecting the user to another step in the assistant.

For example, this is where you would update a user's account based on data the user has entered in the assistant.

The following flowchart provides an overview of a suggested assistant design.



## Using Redirection in an Assistant Workflow

From within a custom assistant you can redirect users to:

- a new record/page within NetSuite (for example, to a new Employee or Contact page in NetSuite)
- the start page of a built-in NetSuite assistant (for example, the Import Assistant or the Web Site Assistant)
- another custom assistant

To link users to another NetSuite page, built-in assistant, or custom assistant, and then return them back to the originating assistant, you must set the value of the customwhence parameter in the redirect URL to originating assistant. The value of customwhence will consist of the scriptId and deploymentId of the originating custom assistant Suitelet.

### Example

The following sample shows a helper function that appears at the end of the Assistant code sample (see [UI Object Assistant Code Sample](#)). Notice that in this function, the value of the customwhence parameter in the URL is the scriptId and deploymentId of the custom assistant that you originally started with. To link users out of the originating assistant, and then return them back to this assistant after they have completed other tasks, you must append the customwhence parameter to the URL you are redirecting to.

```
function getLinkoutURL( redirect, type )
{
    var url = redirect;
    if ( type == "record" )
        url = nlapiResolveURL('record', redirect);
    url += url.indexOf('?') == -1 ? '?' : '&';
    var context = nlapiGetContext();
    url += 'customwhence='+ escape(nlapiResolveURL('suitelet',context.getscriptId(),
        context.getDeploymentId()))
    return url;
}
```



**Note:** If you redirect users to a built-in assistant or to another custom assistant, be aware that they will not see the “Finish” page on the assistant they have been linked out to. After they complete the assistant they have been linked to, they will be redirected back to the page where they left off in the original assistant.

## Assistant Components and Concepts

The following information pertains to the UI components used to build an assistant. Also described are the concepts associated with state management and error handling.

- Steps
- Field Groups
- Fields
- Sublists

- Buttons
- State Management
- Error Handling

## Steps

Create a step by calling `nlobjAssistant.addStep(name, label)`, which returns a reference to the `nlobjAssistantStep` object.

At a minimum, every assistant will include steps, since steps are what define each page of the assistant. Whether the steps must be completed sequentially or in a more random order is up to you. Enforced sequencing of steps will be defined by the `nlobjAssistant.setOrdered(ordered)` method.

The placement of your steps (vertically along the left panel, or horizontally across the top of the assistant) will also be determined by you. Also note that you can add helper text for each step using the `nlobjAssistantStep.setHelpText(help)` method.

 **Note:** Currently there is no support for sub-steps.

## Field Groups

Create a field group by calling `nlobjAssistant.addFieldGroup(name, label)`, which returns a reference to the `nlobjFieldGroup` object.

In the UI, field group are collapsible groups of fields that can be displayed in a one-column or two-column format. The following snippet shows how to use the `nlobjField.setLayoutType(type, breaktype)` method to start a second column in a field group.

```
assistant.addFieldGroup("companyinfo", "Company Information");
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");
```

Note that field groups do not have to be collapsible. They can appear as a static grouping of fields. See `nlobjFieldGroup.setCollapsible(collapsible, hidden)` for more information about setting collapsibility.

## Fields

Create a field by calling `nlobjAssistant.addField(name, type, label, source, group)`, which returns a reference to the `nlobjField` object.

Fields are added to the **current** step on a per-request basis. For example, as the sample below shows, in a GET request, if the user's current step is a step called "companyinformation", (meaning the user has navigated to a step/page with the internal ID "companyinformation"), the page that renders will include a field group and six fields within the group.

```
var step = assistant.getCurrentStep();
if (step.getName() == "companyinformation")
{
```

```

assistant.addFieldGroup("companyinfo", "Company Information");
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");
}

```

Note that all `nlobjField` APIs can be used with the fields returned from the `addField(...)` method. Also, fields marked as 'mandatory' are respected by the assistant. Users cannot click through to the next page if mandatory fields on the current page do not contain a value.



**Important:** The `nlobjField.setLayoutType(type, break)` method can be used to place a column break in an assistant. Be aware that only the first column break that is encountered will be honored. Currently assistants support only single or two column layouts. You cannot set more than one column break.

## Sublists

Create a sublist by calling `nlobjAssistant.addSubList(name, type, label)`, which returns a reference to the `nlobjSubList` object.

If you want to add a sublist to an assistant, be aware that only sublists of type `inlineeditor` are supported. Also note that sublists on an assistant are always placed below all other elements on the page.

## Buttons

You do not need to programmatically add button objects to an assistant. Buttons are automatically generated through the `nlobjAssistant` object.

Depending on which page you are on, the following buttons appear: Next, Back, Cancel, Finish. When users reach the final step in an assistant, the Next button no longer displays, and the Finish button appears. Button actions need to be communicated via the request using `nlobjAssistant.getLastAction()`.



**Important:** The addition of custom buttons are not currently supported on assistants.

## State Management

Assistants support data and state tracking across pages within the **same** session until the assistant is completed by the user (at which point the assistant is reset when the "Finished" page displays).

Field data tracking is automatically saved in assistants. For example, if a user revisits a page using the Back button, the previously entered data will be automatically displayed.

Every time a page is submitted, all the fields will be automatically tracked and when the page is displayed. If the user did not explicitly set a value for a field or on a sublist, then the field(s) and sublist(s) will be populated from data entered by the user the last time they submitted that page.



**Note:** If state/data tracking needs to be preserved across sessions, you should use custom records or tables to record this information.

Note that an SSS\_NOT\_YET\_SUPPORTED\_ERROR is thrown if the assistant is used on an “Available Without Login” (external) Suitelet. (See [Setting Available Without Login](#) for information on this Suitelet deployment option.) Session-based state tracking used in custom assistants requires a session to exist across requests.

Finally, multiple Suitelet deployments should **not** be used to manage the pages within an assistant, since data/state tracking is tied to the Suitelet instance. Developers should create one Suitelet deployment per assistant.

## Error Handling

If an error occurs on a step, the assistant displays two error indicators. The first indicator is a red bar that appears directly below the step. The second indicator is the html you pass to nlobjAssistant.setError(html).

## UI Object Assistant Code Sample

The following is an implementation of a setup assistant with a few basic steps. State is managed throughout the life of the user's session. In summary, this script shows you how to:

1. Create the assistant.
2. Create steps.
3. Set the user's first step.
4. Build pages for each step.
5. Process data entered by the user.

Note that this sample can be run in a NetSuite account. To do so, you must create a .js file for the sample code below. Then you must create a new Suitelet script record and a script deployment. Do not select the Available Without Login deployment option on the Script Deployment page, otherwise the Suitelet will not run. (For general details on creating a Script record, setting values on the Script Deployment page, and creating a custom menu link for a Suitelet, see [Running a Suitelet in NetSuite](#).)

After the script is deployed, you can launch the assistant Suitelet by clicking the Suitelet URL on the Script Deployment page. You can also create a tasklink for the Suitelet and launch the Suitelet as a custom menu item.

Also notice that this sample has all three types of “link outs,” as defined in [Using Redirection in an Assistant Workflow](#): one link out to add employees, one to the Import Assistant, and one to another custom assistant. Notice how the **customwhence** parameter is constructed and appended to the target URL that you are linking out to. Also notice how nlobjAssistant.sendRedirect(response) is used to ensure that customwhence is respected.



**Important:** If your browser is inserting scroll bars in this code sample, maximize your browser window, or expand the main frame that this sample appears in.

```
/*
 * Implementation of a simple setup assistant with support for multiple setup steps and sequential or ad-hoc step traversal.
 * State is managed throughout the life of the user's session but is not persisted across sessions. Doing so would
```

```

/* require writing this information to a custom record.
*/
/* @param request request object
 * @param response response object
*/
function showAssistant(request, response)
{
    /* first create assistant object and define its steps. */
    var assistant = nlapiCreateAssistant("Small Business Setup Assistant", true);
    assistant.setOrdered( true );
    nlapiLogExecution( 'DEBUG', "Create Assistant ", "Assistant Created" );

    assistant.addStep('companyinformation', 'Setup Company Information').setHelpText("Setup your
        <b>important</b> company information in the fields below.");
    assistant.addStep('companypreferences', 'Setup Company Preferences').setHelpText("Setup your
        <b>important</b> company preferences in the fields below.");
    assistant.addStep('enterlocations', 'Enter Locations').setHelpText("Add Locations to your ac
count.

    You can create a location record for each of your company's locations. Then you can track
employees
        and transactions by location..");
    assistant.addStep('enteremployees', 'Enter Company Employees').setHelpText("Enter your
        company employees.");
    assistant.addStep('importrecords', 'Import Records').setHelpText("Import your initial compan
y data.");
    assistant.addStep('configurepricing', 'Configure Pricing' ).setHelpText("Configure your item
pricing.");
    assistant.addStep('summary', 'Summary Information').setHelpText("Summary of your Assistant
Work.<br> You have made the following choices to configure your NetSuite account.");

/* handle page load (GET) requests.*/
if (request.getMethod() == 'GET')
{
    /*.Check whether the assistant is finished */
    if ( !assistant.isFinished() )
    {
        // If initial step, set the Splash page and set the intial step
        if ( assistant.getCurrentStep() == null )
        {
            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.");
        }
        var step = assistant.getCurrentStep();

        // Build the page for a step by adding fields, field groups, and sublists to the assis
tant
    }
}

```

```

if (step.getName() == "companyinformation")
{
    assistant.addField('orgtypelabel','label','What type of organization are
        you?').setLayoutType('startrow');
    assistant.addField('orgtype', 'radio','Business To Consumer',
        'b2c').setLayoutType('midrow');
    assistant.addField('orgtype', 'radio','Business To Business','b2b').setLayoutType(
        'midrow');
    assistant.addField('orgtype', 'radio','Non-Profit','nonprofit').setLayoutType('endr
        ow');
    assistant.getField( 'orgtype', 'b2b' ).setDefaultValue( 'b2b' );

    assistant.addField("companysize","label","How big is your organization?");
    assistant.addField("companysize", 'radio','Small (0-99 employees)', 's');
    assistant.addField("companysize", 'radio','Medium (100-999 employees)', 'm');
    assistant.addField("companysize", 'radio','Large (1000+ employees)', 'l');

    assistant.addFieldGroup("companyinfo", "Company Information");
    assistant.addField("companyname", "text", "Company Name", null,
        "companyinfo").setMandatory( true );
    assistant.addField("legalname", "text", "Legal Name", null, "companyinfo").setManda
    tory
        ( true );
    assistant.addField("shiptoattention", "text", "Ship To Attention", null,
        "companyinfo").setMandatory( true );
    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"
    );
    assistant.addFieldGroup("taxinfo", "Tax Information").setCollapsible(true /* collap
        sable */,
        true /* collapsed by default */);
    assistant.addField("employeeidnumber", "text", "Employee Identification Number (EIN
        )",
        null, "taxinfo").setHelpText("Enter the EID provided to you by the state or
        federal government");
    assistant.addField("taxidnumber", "text", "Tax ID Number", null,
        "taxinfo").setHelpText("Enter the Tax ID number used when you file your payroll
        and sales taxes");
    assistant.addField("returnmailaddress", "textarea", "Return Mail Address", null,
        "taxinfo").setHelpText("In the rare event someone returns your products, enter
        the mailing address.");
}

if (step.getName() == "companypreferences")
{
    nlapiLogExecution( 'DEBUG', "Company Preferences ", "Begin Creating Page" );
}

```

```

assistant.addFieldGroup("companyprefs", "Company Preferences");
var firstDayOfWeek = assistant.addField("firstdayofweek", "select", "First Day of Week",
null, "companyprefs");
var stateAbbrs = assistant.addField("abbreviatestates", "checkbox", "Use State Abbreviations
in Addresses", null, "companyprefs");
var customerMessage = assistant.addField("customerwelcomemessage", "text", "Customer
Center Welcome Message", null, "companyprefs");
customerMessage.setMandatory( true );

assistant.addFieldGroup("accountingprefs", "Accounting Preferences").setCollapsible
(true,
true );
var accountNumbers = assistant.addField("accountnumbers", "checkbox", "Use Account
Numbers", null, "accountingprefs");
var creditLimitDays = assistant.addField("credlimdays", "integer", "Days Overdue
for Warning or Hold", null, "accountingprefs");
var expenseAccount = assistant.addField("expenseaccount", "select", "Default
Expense Account", 'account', "accountingprefs");
customerMessage.setMandatory( true );

assistant.addField('customertypelabel','label','Please Indicate Your Default Custom
er
Type?' );
assistant.addField( 'customertype', 'radio', 'Individual', 'i' );
assistant.addField('customertype', 'radio', 'Company', 'c' );

// get the select options for First Day of Week
nlapiLogExecution( 'DEBUG', "Load Configuration ", "Company Preferences" );
var compPrefs = nlapiLoadConfiguration( 'companypreferences' );

var firstDay = compPrefs.getField( 'FIRSTDAYOFWEEK' );
nlapiLogExecution( 'DEBUG', "Create Day of Week Field ", compPrefs.getFieldText(
'FIRSTDAYOFWEEK' ) );

try
{
    var selectOptions = firstDay.getSelectOptions();
}
catch( error )
{
    assistant.setError( error );
}

if( selectOptions != null)
{
    nlapiLogExecution( 'DEBUG', "Have Select Options ", selectOptions[0].getText()
);

    // add the options to the UI field
    for (var i = 0; i < selectOptions.length; i++)
}

```

```

    {
        firstDayOfWeek.addSelectOption( selectOptions[i].getId(),
            selectOptions[i].getText() );
    }
}

// set the default values based on the product default
stateAbbrs.setDefaultValue( compPrefs.getFieldValue( 'ABBREVIATESTATES' ) );
customerMessage.setDefaultValue( compPrefs.getFieldValue(
    'CUSTOMERWELCOMEMESSAGE' ) );

}

else if (step.getName() == "enterlocations")
{
    var sublist = assistant.addSubList("locations", "inlineeditor", "Locations");

    sublist.addField("name", "text", "Name");
    sublist.addField("tranprefix", "text", "Transaction Prefix");
    sublist.addField("makeinventoryavailable", "checkbox", "Make Inventory Available");

    sublist.addField("makeinventoryavailablestore", "checkbox", "Make Inventory Available in
le in
Web Store");

    sublist.setUniqueField("name");
}

else if (step.getName() == "enteremployees")
{
    // get the host
    var host = request.getURL().substring(0, ( request.getURL().indexOf('.com') + 4 ) );

    assistant.addFieldGroup("enteremps", "Enter Employees");
    assistant.addField("employeecount", "integer", "Number of Employees in Company", nu
ll,
    "enteremps").setMandatory( true );
    assistant.addField("enterempslink", "url", "", null, "enteremps" ).setDisplayType
        ( "inline" ).setLinkText( "Click Here to Enter Your Employees").setDefaultValue(
host
    + getLinkoutURL( 'employee', 'record' ) );
}

else if (step.getName() == "importrecords")
{
    var host = request.getURL().substring(0, ( request.getURL().indexOf('.com') + 4 ) );

    assistant.addFieldGroup("recordimport", "Import Data");
    assistant.addField("recordcount", "integer", "Number of Records to Import", null,
        "recordimport").setMandatory( true );
    assistant.addField("importlink", "url", "", null, "recordimport" ).setDisplayType
        ( "inline" ).setLinkText( "Click Here to Import Your Data").setDefaultValue( host +

```

```

        getLinkoutURL( "/app/setup/assistants/nsimport/importassistant.nl" ) );
    }

    else if (step.getName() == "configurepricing")
    {
        var host = request.getURL().substring(0, ( request.getURL().indexOf('.com') + 4 ) );

        assistant.addFieldGroup("pricing", "Price Configuration");
        assistant.addField("itemcount", "integer", "Number of Items to Configure", null,
            "pricing").setMandatory( true );

        /* When users click the 'Click Here to Configure Pricing' link, they will be taken to
         * another custom assistant Suitelet that has a script ID of 47 and a deploy ID of 1. Note
         * that the code for the "link out" assistant is not provided in this sample.
         *
         * Notice the use of the getLinkoutURL helper function, which sets the URL
         * customwhence parameter so that after users finish the with the "link out" assistant,
         * they will be redirected back to this (the originating) assistant.
        */
        assistant.addField("importlink", "url", "", null, "pricing").setDisplayType
            ( "inline" ).setLinkText( "Click Here to Configure Pricing" ).setDefaultValue( hos
        t +
            getLinkoutURL( "/app/site/hosting/scriptlet.nl?script=47&deploy=1" ) );
    }

    else if (step.getName() == "summary")
    {

        assistant.addFieldGroup("companysummary", "Company Definition Summary");
        assistant.addField('orgtypelabel','label','What type of organization are you?', nul
        l,
            'companysummary' );
        assistant.addField('orgtype', 'radio','Business To Consumer', 'b2c',
            'companysummary' ).setDisplayType( 'inline' );
        assistant.addField('orgtype', 'radio','Business To Business', 'b2b',
            'companysummary' ).setDisplayType( 'inline' );
        assistant.addField('orgtype', 'radio','Non-Profit', 'nonprofit',
            'companysummary' ).setDisplayType( 'inline' );

        assistant.addField('companysizelabel','label','How big is your organization?', null
        ,
            'companysummary' );
        assistant.addField('companysize', 'radio','Small (0-99 employees)', 's',
            'companysummary' ).setDisplayType( 'inline' );
        assistant.addField('companysize', 'radio','Medium (100-999 employees)', 'm',
            'companysummary' ).setDisplayType( 'inline' );
        assistant.addField('companysize', 'radio','Large (1000+ employees)', 'l',
            'companysummary' ).setDisplayType( 'inline' );
    }
}

```

```

        'companysummary' ).setDisplayType( 'inline') );
    assistant.addField("companynam", "text", "Company Name", null,
        "companysummary").setDisplayType( 'inline');
    assistant.addField("city", "text", "City", null, "companysummary").setDisplayType('
    inline');
    assistant.addField("abbreviatestates", "checkbox", "Use State Abbreviations in Addr
    esses",
        null, "companysummary").setDisplayType( 'inline');
    assistant.addField("customerwelcomemessage", "text", "Customer Center Welcome
    Message", null, "companysummary").setDisplayType( 'inline');

    // get previously submitted steps
    var ciStep = assistant.getStep( 'companyinformation' );
    var cpStep = assistant.getStep( 'companypreferences' );

    // get field values from previously submitted steps
    assistant.getField( 'orgtype', 'b2b' ).setDefaultValue( ciStep.getFieldValue( 'orgt
    ype' ) );
    assistant.getField( 'companysize', 's' ).setDefaultValue( ciStep.getFieldValue
        ( 'companysize' ) );
    assistant.getField( 'companynam' ).setDefaultValue( ciStep.getFieldValue
        ( 'companynam' ) );
    assistant.getField( 'city' ).setDefaultValue( ciStep.getFieldValue( 'city' ) );
    assistant.getField( 'abbreviatestates' ).setDefaultValue( cpStep.getFieldValue
        ( 'abbreviatestates' ) );
    assistant.getField( 'customerwelcomemessage' ).setDefaultValue
        ( cpStep.getFieldValue( 'customerwelcomemessage' ) );

    }

}

response.writePage(assistant);
}
/* handle user submit (POST) requests. */
else
{
    assistant.setError( null );

    /* 1. if they clicked the finish button, mark setup as done and redirect to assistant pag
    e */
    if (assistant.getLastAction() == "finish")
    {
        assistant.setFinished( "You have completed the Small Business Setup Assistant." );

        assistant.sendRedirect( response );
    }
    /* 2. if they clicked the "cancel" button, take them to a different page (setup tab) alto
    gether as
        appropriate. */
    else if (assistant.getLastAction() == "cancel")
    {

```

```

        nlapiSetRedirectURL('tasklink', "CARD_-10");
    }
    /* 3. For all other actions (next, back, jump), process the step and redirect to assistant
       page. */
    else
    {

        if (assistant.getLastStep().getName() == "companyinformation" && assistant.getLastAction()
on()
            == "next" )
        {
            // update the company information page
            var configComplInfo = nlapiLoadConfiguration( 'companyinformation' );

            configComplInfo.setFieldValue( 'city', request.getParameter( 'city' ) ) ;

            nlapiSubmitConfiguration( configComplInfo );
        }

        if (assistant.getLastStep().getName() == "companypreferences" && assistant.getLastAction()
on()
            == "next" )
        {
            // update the company preferences page
            var configCompPref = nlapiLoadConfiguration( 'companypreferences' );

            configCompPref.setFieldValue( 'CUSTOMERWELCOMEMESSAGE',
                request.getParameter( 'customerwelcomemessage' ) );

            nlapiSubmitConfiguration( configCompPref );

            // update the accounting preferences pages
            var configAcctPref = nlapiLoadConfiguration( 'accountingpreferences' );

            configAcctPref.setFieldValue( 'CREDLIMDAYS', request.getParameter( 'credlimdays' ) );
        });

        nlapiSubmitConfiguration( configAcctPref );
    }

    if (assistant.getLastStep().getName() == "enterlocations" && assistant.getLastAction()
== "next" )
    {
        // create locations

        for (var i = 1; i <= request.getLineItemCount( 'locations'); i++)
        {
            locationRec = nlapiCreateRecord( 'location' );

            locationRec.setFieldValue( 'name', request.getLineItemValue( 'locations', 'name'
, i ) );
            locationRec.setFieldValue( 'tranprefix', request.getLineItemValue( 'locations',
                'tranprefix', i ) );
            locationRec.setFieldValue( 'makeinventoryavailable', request.getLineItemValue

```

```
('locations', 'makeinventoryavailable', i ) );
locationRec.setFieldValue( 'makeinventoryavailablestore',
    request.getLineItemValue('locations', 'makeinventoryavailablestore', i ) );

try
{
    // add a location to the account
    nlapiSubmitRecord( locationRec );
}
catch( error )
{
    assistant.setError( error );
}

if( !assistant.hasError() )
    assistant.setCurrentStep( assistant.getNextStep() );

assistant.sendRedirect( response );

}

}

function getLinkoutURL( redirect, type )
{
    var url = redirect;

    if ( type == "record" )
        url = nlapiResolveURL('record', redirect);

    url += url.indexOf('?') == -1 ? '?' : '&';

    var context = nlapiGetContext();
    url += 'customwhence=' + escape(nlapiResolveURL('suitelet',context.getScriptId(), context.getDeploymentId()));

    return url;
}
```

# Working with the SuiteScript Debugger



**Note:** The content in this help topic pertains to all versions of SuiteScript.

- SuiteScript Debugger Overview
- Using the SuiteScript Debugger

## SuiteScript Debugger Overview



**Note:** The content in this help topic pertains to all versions of SuiteScript.

You can use the SuiteScript Debugger to debug server-side scripts and core plug-in implementations.

To debug **client** scripts, NetSuite recommends using the Chrome DevTools for Chrome, the Firebug debugger for Firefox, and the Microsoft Script Debugger for Internet Explorer. For additional information about these tools, see the documentation provided with each browser.



**Note:** If you use the SuiteCloud IDE to develop your SuiteScripts, see the help topic [Using the SuiteCloud IDE Debugger](#). The SuiteCloud IDE Debugger enables you to debug server-side SuiteScripts in the same Eclipse-based environment you develop them in.

To work with the SuiteScript Debugger, you will need to learn about the following:

1. Accessing a **Debugger** domain as well as requirements for running the SuiteScript Debugger (see [Before Using the SuiteScript Debugger](#))
2. SuiteScript Debugger metering and permission restrictions (see [SuiteScript Debugger Metering and Permissions](#))
3. How to use the SuiteScript Debugger (see [Using the SuiteScript Debugger](#))
4. SuiteScript Debugger tabs and buttons (see [SuiteScript Debugger Interface](#))

To view script execution details either while using the SuiteScript Debugger, or after the script has been deployed into NetSuite, see [Creating Script Execution Logs](#).

## Using the SuiteScript Debugger



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The SuiteScript Debugger provides two debugging modes, which are based on the type of script you want to debug.

- **Ad Hoc Debugging:** Enables you to debug code fragments written “on-the-fly.” With ad-hoc debugging you are debugging a new script or code snippet that does not have a defined SuiteScript deployment. Scripts that do not require any form/record-specific interaction are good candidates for ad-hoc debugging.

- **Deployed Debugging:** Enables you to select an existing script or core plug-in implementation that already has a defined Script Deployment or Plug-in Implementation record. The status of your script or implementation must be set to *Testing* before it can be loaded into the Debugger. You must also be the owner.

# Before Using the SuiteScript Debugger

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Before using the SuiteScript Debugger, you must be aware of the following :

- There are three separate Debugger domains, accessible by going to Customization > Scripting > Script Debugger from a production account, a Beta account, or a sandbox account. You can also access a Debugger domain by going directly to the following URLs:
  - <https://debugger.netsuite.com/pages/login.jsp>.

You must then log in to NetSuite.



**Important:** Any changes you make to your account while on this Debugger domain will affect the data in your production account. For example, if you execute a script in the Debugger that creates a new record, that record will appear in your production account.

- <https://debugger.beta.netsuite.com>

If you choose to run the Debugger in the release preview or beta environment, go to this Debugger domain.



**Note:** If you go to Customization > Scripting > Script Debugger in your Beta account, you will be directed to the Debugger Beta domain. Any changes you make to your account while on the Debugger Beta domain will affect your Beta account only.

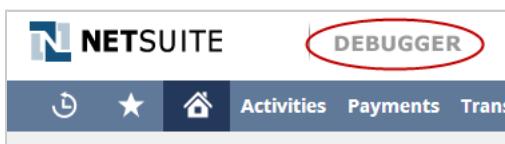
- <https://debugger.sandbox.netsuite.com>

If you have a sandbox account and choose to run the Debugger in this environment, go to this Debugger domain.



**Note:** If you go to Customization > Scripting > Script Debugger in your sandbox account, you will be directed to the Debugger sandbox domain. Any changes you make to your account while on the Debugger sandbox domain will affect your sandbox account only.

When you are logged on to a Debugger domain, you see the Debugger logo at the top of your account.



**Note:** There may be a decrease in performance when working on Debugger domains.

- The Debugger executes server-side scripts only (these script types include **Suitelets**, **Portlet** scripts, **Scheduled** scripts, and **User Event** scripts). Client scripts (both form- and record-level) should be tested on the form/record they run against.
- To debug scripts, the following must apply:
  - You must have scripting permission.
  - You must be the assigned owner of the script.

- If you are debugging a script that already has a defined deployment, that script must be in *Testing* mode before it can be loaded into the Debugger. If you want to debug a script that has already been released into production, you must change the script's status from *Released* to *Testing* on the Script Deployment page.
- If a bundled script has been installed into your account and the script has been marked as *hidden*, you will be unable to debug this script.

Be aware that the SuiteScript Debugger is not any of the following:

- An API test console
- An integrated development environment (IDE)
- A script deployment interface. To deploy SuiteScript to NetSuite, you must still create a script record and define the script's deployment parameters on the Script Deployment page.
- A Client SuiteScript debugger
- A script runner interface (for example, scheduled scripts still need to be put INQUEUE for task completion)

# Ad Hoc Debugging



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Ad hoc debugging is for testing scripts or code snippets that do not have a defined SuiteScript deployment. Ad hoc debugging is **not** for scripts that have a Script record or defined deployment parameters (set on the Script Deployment page).

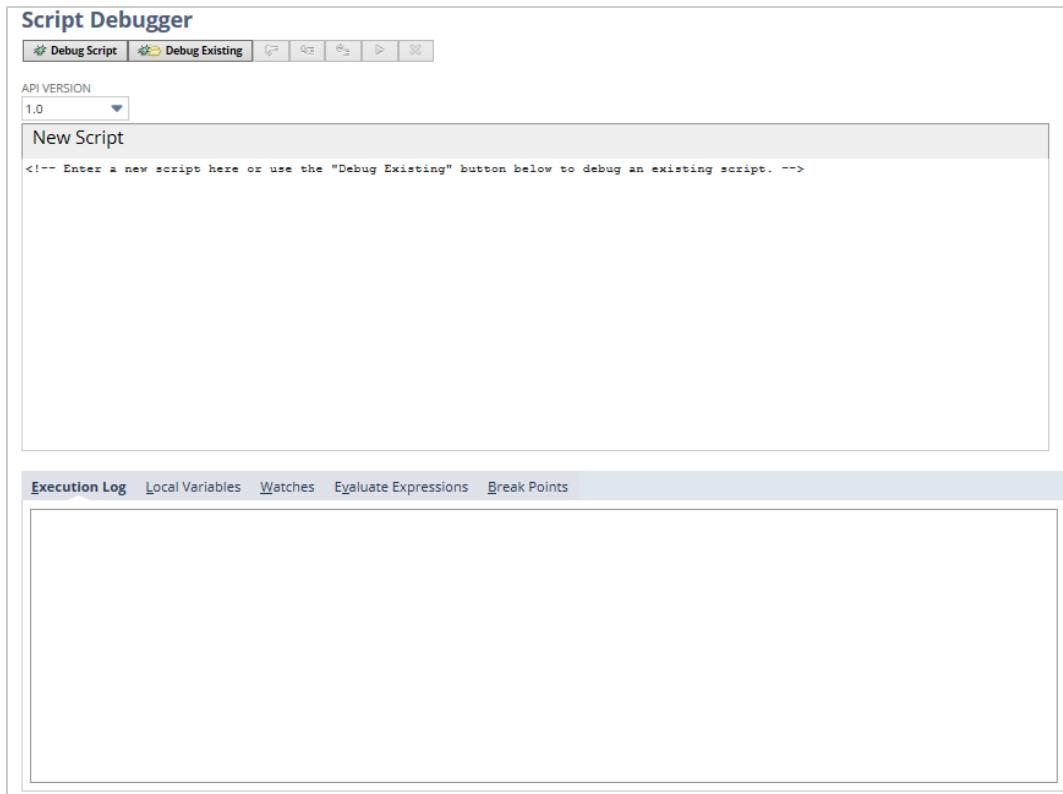
## To use the Debugger in ad hoc mode:

1. Go to Customization > Scripting > Script Debugger if you are already logged in to a production, Beta, or sandbox NetSuite account. Or, go directly to one of the following Debugger domains:
  - <https://debugger.netsuite.com>
  - <https://debugger.beta.netsuite.com> (if you choose to run the Debugger in the release preview or beta environment)
  - <https://debugger.sandbox.netsuite.com> (if you have a sandbox account and choose the run the Debugger in this environment)



**Important:** See [Before Using the SuiteScript Debugger](#) to learn about how the changes you make to your account on a Debugger domain can affect your production, Beta, or sandbox account.

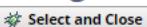
The following figure shows the Script Debugger start page.



2. Type your code snippet in the **New Script** text area.

If you have already written your code in an IDE, copy and paste the code into the Debugger text area. If you modify your script in the Debugger and you intend to save the changes, you must copy the updated script from the Debugger and paste it into your IDE.

**Debug Existing**

 Select and Close

Select a script from the list below and click the "Select and Close" button for it to be loaded into the debugger. For non-scheduled scripts you must first perform an action that invokes the script.

SELECT	TYPE	SCRIPT	TITLE	FUNCTION
<input type="radio"/>	User Event	Set Metadata	SPM_Test_Record	
<input type="radio"/>	User Event	Email Manager script	SPM_Test_Record	
<input type="radio"/>	User Event	Set Metadata	Customer	
<input checked="" type="radio"/>	User Event	Email Manager script	Customer	
<input type="radio"/>	User Event	Set Metadata	Employee	
<input type="radio"/>	User Event	Email Manager script	Employee	
<input type="radio"/>	User Event	Set Metadata	Journal Entry	
<input type="radio"/>	User Event	Email Manager script	Journal Entry	

3. Next, click the **Debug Script** button.

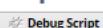
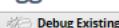
The script is immediately loaded into the Debugger and the program's execution stops before running the first line of executable code.



**Important:** Make sure you always have an executable call in your script; otherwise, the script has nothing to execute and return.

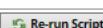
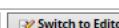
4. With the ad hoc script loaded into the Debugger, you can now step through each line to inspect local variables and object properties. You can also add watches, evaluate expressions, and set break points. See [SuiteScript Debugger Interface](#) for information on stepping into/out of functions, adding watches, setting and removing break points, and evaluating expressions.
5. After testing/debugging your ad hoc script, you can either re-run the script (by clicking the **Re-run Script** button), or put the script into edit mode (by clicking the **Switch to Editor** button) and continue debugging.

**Script Debugger**

API VERSION  
1.0

Completed New Script Execution

# Deployed Debugging



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Deployed-mode debugging is for testing and inspecting scripts or core plug-in implementations that have a defined Script Deployment or Plug-in Implementation record.



**Note:** SuiteScript does not support read-only sublists. If you are debugging a script that loads a record with a custom child record as a sublist, make sure the **Allow Child Record Editing** setting is checked for the child record in SuiteBuilder. If this box is not checked, the sublist is read-only and will not load in the parent record. See the help topic [Creating Custom Record Types](#) for additional information on creating custom records.

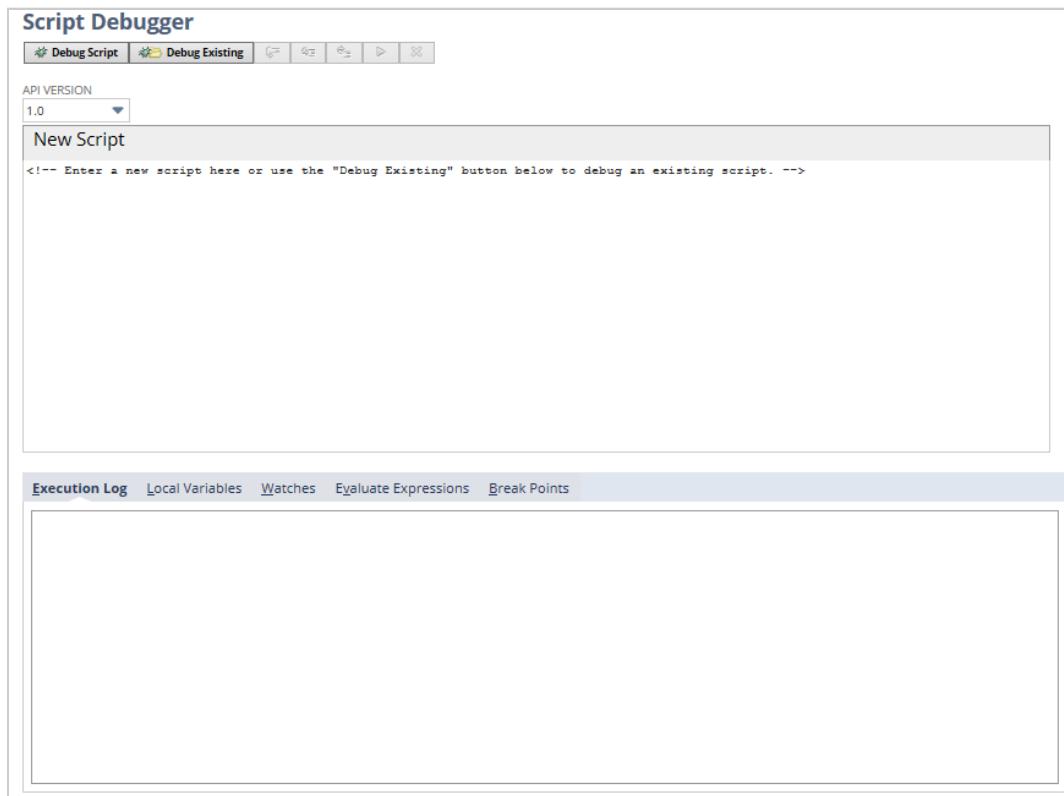
## To use the Debugger in deployed-mode:

1. Go to Customization > Scripting > Script Debugger if you are already logged in to a production, Beta, or sandbox NetSuite account. Or, go directly to one of the following Debugger domains:
  - <https://debugger.netsuite.com>
  - <https://debugger.beta.netsuite.com> (if you choose to run the Debugger in the release preview or beta environment)
  - <https://debugger.sandbox.netsuite.com> (if you have a sandbox account and choose to run the Debugger in this environment)
2. When you are on a Debugger domain, navigate to the applicable Script Deployment or Plug-in Implementation record and verify that **Status** is set to **Testing**. If you wish to debug a script or implementation that has already been released into production, you must change the status from **Released** to **Testing**.

Script Deployment		List Search More	
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>			
SCRIPT	Update Item Price	STATUS *	Testing
APPLIES TO *	Sales Order	EVENT TYPE	
ID		LOG LEVEL	Debug
<input checked="" type="checkbox"/> DEPLOYED		EXECUTE AS ROLE	Administrator

3. Verify that you are the assigned owner of the script or implementation. You can only debug if you are the assigned owner.
4. To get to the Debugger start page, use the appropriate option:
  - If you are on the Script Deployment page in View mode, click the **Debug** button. If you are on the Script Deployment page in Edit mode, click the **Save and Debug** button.

- If you are not on the Script Deployment page or you are debugging an implementation, you can navigate to the Debugger by going to Customization > Scripting > Script Debugger.
5. When the Script Debugger page opens, click the **Debug Existing** button.



After clicking **Debug Existing**, the Script Debugger popup window opens. The popup shows all the server-side scripts and core plug-in implementations that are available for debugging (see figure below). Note that only scripts and implementations whose statuses are set to **Testing** will appear in the Script Debugger popup.

6. Select the script you want to debug, and then click the **Select and Close** button in the popup window.

The following figure shows the selection of a `beforeLoad` User Event script that runs on Customer records. When a new Customer is created, this script sets a custom check box field called **Export to OpenAir** to true.

**Debug Existing**

**Select and Close**

Select a script from the list below and click the "Select and Close" button for it to be loaded into the debugger. For non-scheduled scripts you must first perform an action that invokes the script.

SELECT	TYPE	SCRIPT	TITLE	FUNCTION
<input type="radio"/>	User Event	Set Metadata	SPM_Test_Record	
<input type="radio"/>	User Event	Email Manager script	SPM_Test_Record	
<input type="radio"/>	User Event	Set Metadata	Customer	
<input checked="" type="radio"/>	User Event	Email Manager script	Customer	
<input type="radio"/>	User Event	Set Metadata	Employee	
<input type="radio"/>	User Event	Email Manager script	Employee	
<input type="radio"/>	User Event	Set Metadata	Journal Entry	
<input type="radio"/>	User Event	Email Manager script	Journal Entry	

- After clicking **Select and Close**, the Waiting for User Action screen appears. To load a User Event, Portlet, Suitelet, or Core Plug-in Implementations into the Debugger, you must perform the action that executes its logic.

**Note:** You do not have to perform any user actions for Scheduled Scripts to load into the Debugger. Simply select your Scheduled Script from the Script Debugger popup window, and then click **Save and Close**. The Scheduled Script automatically loads.

In this example, you must create a new Customer for the User Event script to load into the Debugger. Performing the action that actually calls the script creates a “real-time,” live context for script debugging.

**Script Debugger**

**Debug Script** **Debug Existing**

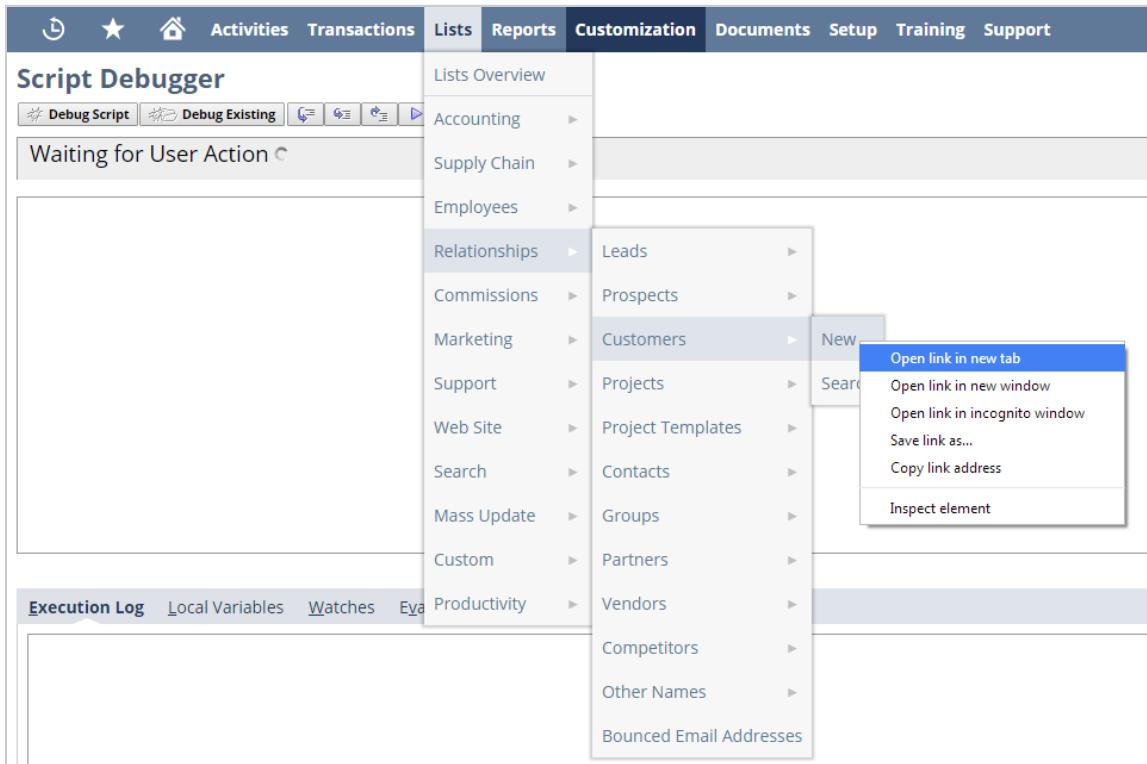
Waiting for User Action

- Next, create a new Customer to load the User Event script into the Debugger.



**Important:** It is highly recommended that when performing user actions, you do so in another window or another tab (see figure below). Doing so enables you to keep the Debugger window open so that you can see the script or implementation as it loads.

- The first figure shows that a new Customer will be created in a separate window.
- The second figure shows the code in the script file as it loads into the Script Debugger window.



```

1 function SNBeforeSubmit()
2 {
3     sleep(500);
4     nlapiLogExecution('DEBUG', 'Before Submit - Slept for 0.5 second');
5 }
6
7 function sleep(milliseconds)
8 {
9     var start = new Date().getTime();
10    for (var i = 0; i < 1e7; i++)
11    {
12        if ((new Date().getTime() - start) > milliseconds)
13        {
14            break;
15        }
16    }
17 }
18
19
20
21 SNBeforeSubmit(type)

```

The code in the text area is as follows:

```

function SNBeforeSubmit()
{
    sleep(500);
    nlapiLogExecution('DEBUG', 'Before Submit - Slept for 0.5 second');

}

function sleep(milliseconds)
{
    var start = new Date().getTime();
    for (var i = 0; i < 1e7; i++)
    {
        if ((new Date().getTime() - start) > milliseconds)
        {
            break;
        }
    }
}

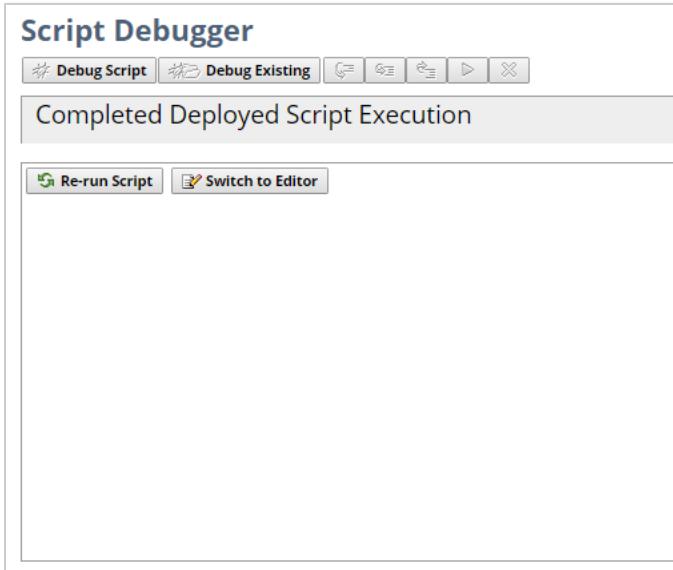
SNBeforeSubmit(type)

```

With the code in the Debugger text area, you have the following options:

- Click the Step Over button to begin stepping through each line of code.
- Add watches and evaluate expressions.
- Set break points and click the Run button to run the code. The Debugger will stop code execution at the first break point set.
- Set no break points, click the Run button, and have the Debugger execute the entire piece of code.

After testing/debugging a deployed-mode script or implementation, you can either re-run it (by clicking the Re-run Script button), or put it into edit mode (by clicking the Switch to Editor button) and continue debugging.



**Important:** If you modify a script or implementation in the Debugger, and you intend to save the changes, you must copy the updated script from the Debugger and paste it into your IDE. You must then re-load the updated .js file into the NetSuite file cabinet.

# SuiteScript Debugger Interface

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

See the following sections to learn about the Debugger UI.

- [SuiteScript Debugger Buttons](#)
- [SuiteScript Debugger Tabs](#)

Use the Debugger buttons to control the flow of script execution. Use the tabs to inspect all script variables, objects, and properties.

## SuiteScript Debugger Buttons

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

 Line 19 is highlighted with a yellow background, indicating it is the current line of execution. A green circle with a checkmark is placed next to the line number 19, indicating a breakpoint has been set at that line.
 

There are five Debugger buttons that can be used to control/resume script execution when the Debugger stops at a particular line:

**Note:** Keyboard shortcuts are enabled for all five Debugger buttons. See [SuiteScript Debugger Keyboard Shortcuts](#) for details.

Icon	Name	Description
	Step Over	Resumes execution from the current line and stops at the next line (even if the current line is a function call).
	Step Into	Resumes execution from the current line and stops at the first line in any function call made from the current line.
	Step Out	Resumes execution from the current line until the end of the current function, and stops at the first line following the line from where this

Icon	Name	Description
		function was called -or- until the next break point -or- until the program terminates (either by error or by normal completion).
▶	Continue	Resumes program execution from the current line until the next break point -or- until the program terminates.
✖	Cancel	Aborts execution of the program from the current line.

## SuiteScript Debugger Tabs

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The SuiteScript Debugger includes the following tabs. Click the links below to jump to information about each tab.

- [Execution Log](#)
- [Local Variables](#)
- [Watches](#)
- [Evaluate Expressions](#)
- [Break Points](#)

### Execution Log

Click the Execution Log tab to view all the execution logs (including errors logged by the system) created by the currently executing program. The execution log details that appear on this tab are the same details that would normally appear in the Execution Log on the Script Deployment page. However, when working in the Debugger, all script execution details appear on the Execution Log tab within the Debugger; these details will NOT appear on the Execution Log tab on the Script Deployment page.

The type, subject, details, and timestamp are displayed in the on the Debugger Execution Log tab. The timestamp is recorded on the server but converted to the current user's time zone for display. The console is automatically cleared at the start of every debugging session.

Log details are collapsed by default, but can be seen by clicking the expand/collapse icon.

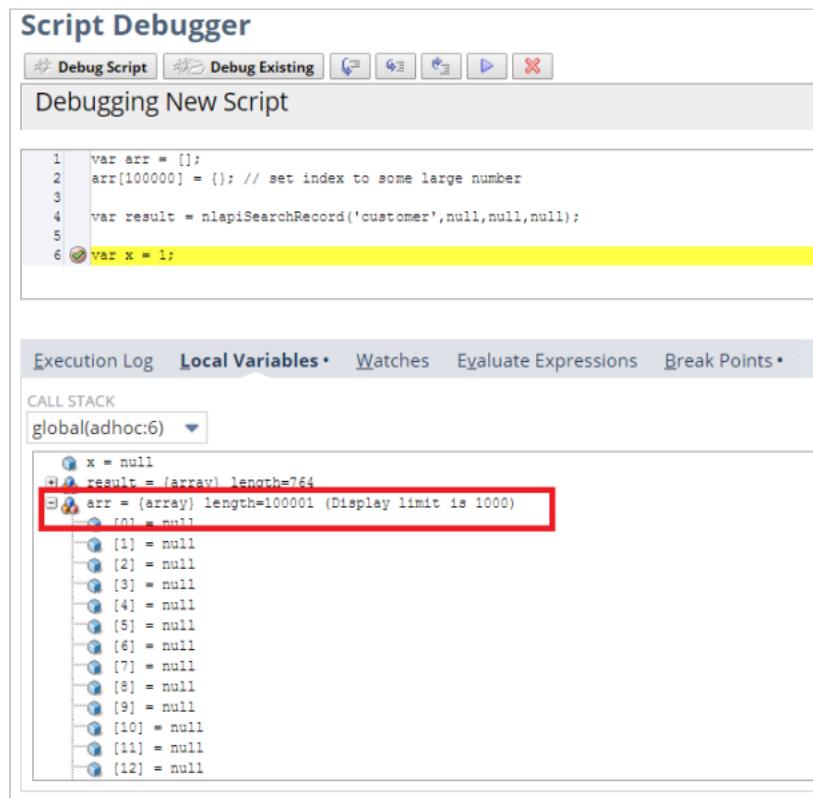
Execution Log • Local Variables   Watches   Evaluate Expressions   Break Points •		
system	INVALID_SSS_DEBUG_SESSION You have cancelled your current script debugging session.	28.8.2014 9:53:15.610
metrics	usage 0, time 22566	28.8.2014 9:53:15.608
debug	i is 12	28.8.2014 9:53:05.289
debug	m is 8	28.8.2014 9:53:05.289
debug	p is 8	28.8.2014 9:53:05.289

### Local Variables

Click the Local Variables to see a browse-able list of all local variables (primitives, objects, and NetSuite objects) currently in scope. Note that for NetSuite (nlobj) objects, all properties are private, even though they can be seen on the Local Variables tab. Do not try to reference these properties directly in your script. Use the appropriate getter/setter functions instead.

Click the Call Stack drop-down to see a list containing a browse-able view of the current execution stack of the program. The function call and current line number for that function are included in the Call Stack drop-down. Use the Call Stack drop-down to switch to different call stacks for local variable observation. In addition, watch expressions and expression evaluations will automatically be performed in the context specified by this field.

**Important:** Due to performance considerations, the member display limit for all variables is 1000. In addition, only the first 500 characters of a String are displayed. For large variables, use a watch to see the full member display (see [Watches](#) for additional information).

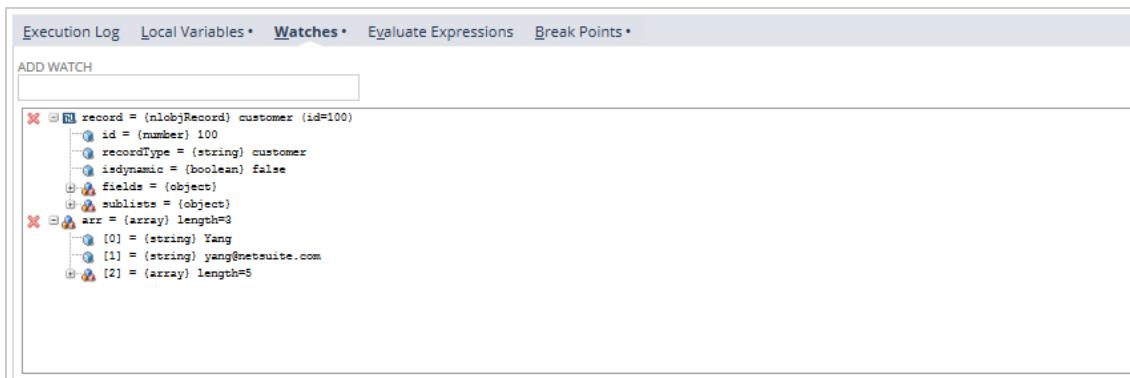


## Watches

The Watch tab is where you can add or remove expressions to a list that is maintained and kept up-to-date ("watched") throughout the execution of a script. The expressions are always evaluated in the current call stack. This means that by default they are evaluated at the current line of script execution. However, if you switch to a different function in the call stack, they will be re-evaluated in that context.

- To add an expression, type it into the Add Watch field and press the Enter key.
- To remove a watch expression, click on the x icon to the left of the expression.
- To browse sub-properties of a user-defined object, an array, or a NetSuite object (nlobj) expression, click the expand/collapse icon next to the property name.

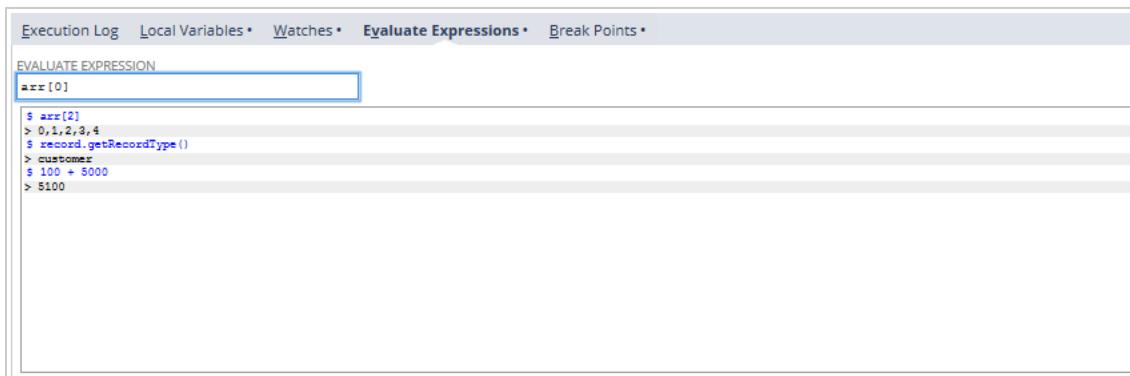
Note that you can use the watch window to navigate to object properties via a command line interface. Any property that is viewable from the property browser can be added as a watch expression by referencing the property using dot (.) notation, even if the property is private in the actual script (for example, you can reference the ID of an nlobjRecord object (referenced by a variable called *record*) by typing *record.id* ).



## Evaluate Expressions

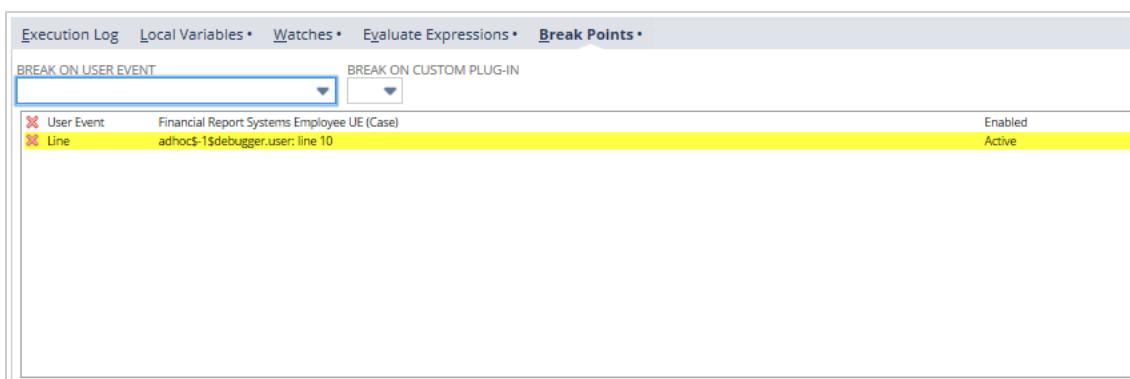
Use the Evaluate Expressions tab to execute code at break points during the current program. Doing so provides access to the program's state, allowing you to modify the state of the program.

Enter an expression in the Evaluate Expression field and press the Enter key to run it at the selected call stack. The results of an evaluated expression (if any) is displayed in the window below. Any changes to the program's state will immediately be reflected in the Local Variables and Watches windows.



## Break Points

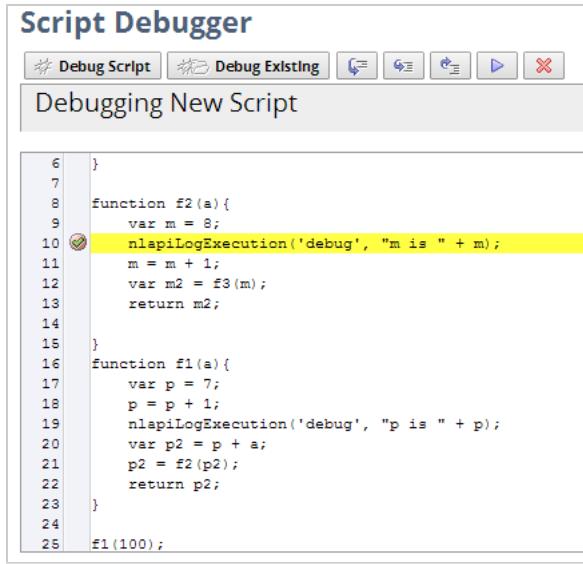
Click the Break Points tab to view all of your instruction-level (line) break points as well as your user event break points (see figure). Note that you can add user event break points by selecting user events from the Break on User Event drop-down.



By setting breakpoints in your code, you can execute your code up to a certain point, and then halt the execution at the break point and examine the current state of the execution.

### To add/remove break points in your code:

1. To add a break point, click between the line number and the actual line of code (see figure below).
2. To remove a break point, click the break point icon as it appears in the code. You can also remove a break point by clicking the x icon next to the break point (as it appears on the **Break Points** tab).



The screenshot shows the 'Script Debugger' window titled 'Debugging New Script'. At the top, there are tabs for 'Debug Script' and 'Debug Existing', along with several icons for debugging operations. The main area displays a block of JavaScript code. On line 10, there is a red circular icon with a green checkmark, indicating a breakpoint has been set. The code itself is as follows:

```

6 }
7
8 function f2(a){
9     var m = 8;
10    nlapiLogExecution('debug', "m is " + m);
11    m = m + 1;
12    var m2 = f3(m);
13    return m2;
14
15 }
16 function f1(a){
17     var p = 7;
18     p = p + 1;
19     nlapiLogExecution('debug', "p is " + p);
20     var p2 = p + a;
21     p2 = f2(p2);
22     return p2;
23 }
24 f1(100);
25

```

# SuiteScript Debugger Metering and Permissions



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The SuiteScript Debugger adheres to the following metering and permission restrictions:

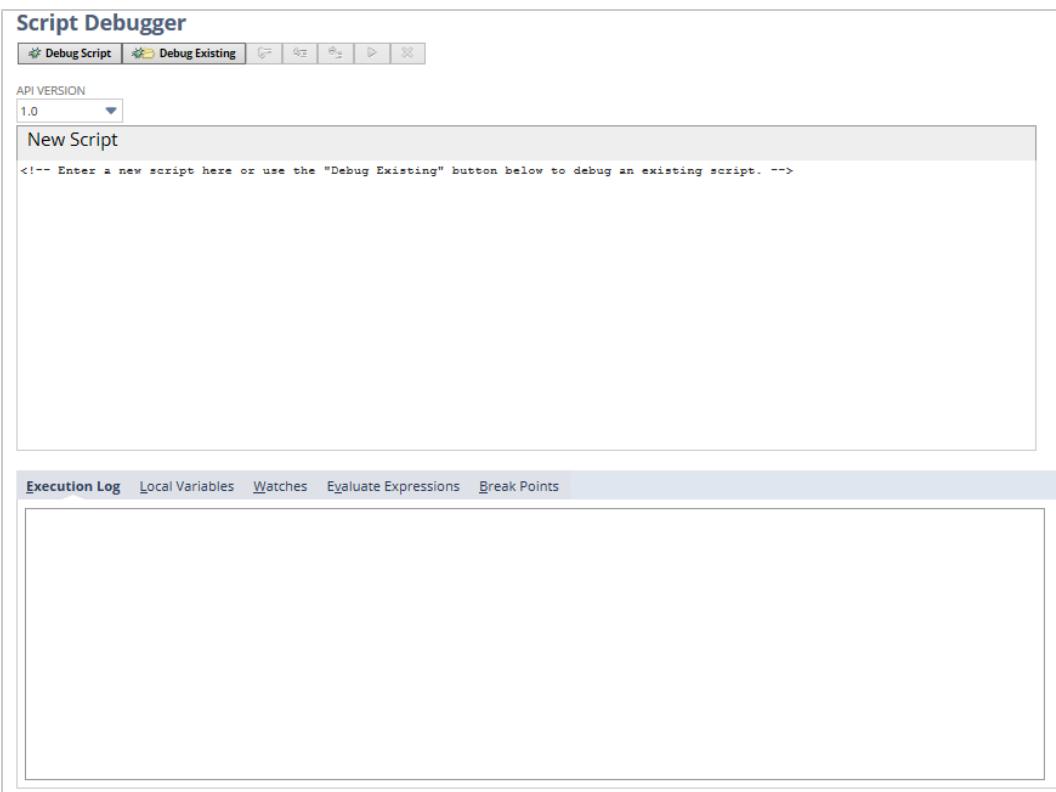
- A user is only allowed to debug a single script at a time. Attempting to debug multiple scripts simultaneously (for example, by opening two different browser windows) will result in the same script/debugging session appearing in both windows.
- Users can debug only their own scripts in their current login session.
- There is a 1000 unit usage limit on all scripts being debugged. This is important to note, particularly for script types such as Scheduled scripts, which are permitted 10,000 units when running in NetSuite. If, for example, you load a 2,000 unit Scheduled script into the Debugger and attempt to step through or execute your code, the Debugger will throw a usage limit error when it reaches 1000 units.
- Email error notification is disabled for scripts being debugged. Additionally, execution log details are displayed on the Execution Log tab in the Debugger rather than in the execution log on the Script Deployment page.
- There is a two minute time limit on scripts sitting idle in the Debugger. If you do not perform some user action in the Debugger within the two minutes, the following error is thrown (see figure):

You have exceeded the maximum allowable idle time for debugging scripts. To debug another script,  
simply reload the script debugger page and start a new debugging session.

If you receive this message and you are in deployed-debugging mode, click **Go Back**. You must then reload your script by clicking **Debug Existing** (see figure below).

If you are debugging in ad-hoc mode, click the Go Back button, click in the Debugger text area, and re-type your code snippet.

(See [Using the SuiteScript Debugger](#) for information on deployed and ad-hoc debugging modes.)



- There is a 10 minute global timeout on all scripts being debugged. This means that even if you are performing user actions within the Debugger every two minutes, the Debugger itself will timeout after 10 minutes.

# SuiteScript Debugger Keyboard Shortcuts



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The following table lists the keyboard shortcuts that are enabled for all five debugger action buttons. Your cursor must be inside the main Debugger text area for the keyboard shortcuts to work.

Key	Command
i	Step Into
space	Step Over
o	Step Out
shift+space	Continue
q	Quit
d	Debug Script (New)
a	Debug Script (Existing)
r	Re-run Script

# SuiteScript Debugger Glossary



**Note:** The content in this help topic pertains to all versions of SuiteScript.

Keyword	Definition
Line Break Point	User-defined line in source code where program halts execution
User Event Break Point	User event possibly invokable during script execution where the program halts execution
Watch	A variable expression that will be monitored throughout the program's execution in the current scope
Call Stack	A stack (most recent on top) of all the active functions (and their local variables) called up until the current line of execution



# SuiteScript 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 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 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 [SuiteScript Reference](#) 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 - 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 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 once.

- 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 sublist on a record – see [Sublist APIs](#)
- Searching in NetSuite – see [Search APIs](#)
- Scheduling scripts to run at specified times – 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
<code>nlapiAttachRecord(type, id, type2, id2, attributes)</code>
<code>nlapiCopyRecord(type, id, initializeValues)</code>
<code>nlapiCreateCSVImport( )</code>

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, baseld, altType, altId, fields)
nlapiMergeTemplate(id, baseType, baseld, altType, altId, fields)
nlapiPrintRecord(type, id, mode, properties)
nlapiSubmitCSVImport(nlobjCSVImport)
nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)
nlapiTransformRecord(type, id, transformType, transformValues)
nlapiVoidTransaction(transactionType, recordId)
<b>Objects</b>
nlobjCSVImport object and all methods
nlobjRecord object and all methods

## Subrecord APIs

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



## Field APIs

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

## Sublist APIs

Functions
<a href="#">nlapiCancelLineItem(type)</a>
<a href="#">nlapiCommitLineItem(type)</a>
<a href="#">nlapiDisableLineItemField(type, fldnam, val)</a>
<a href="#">nlapiFindLineItemMatrixValue(type, fldnam, val, column)</a>
<a href="#">nlapiFindLineItemValue(type, fldnam, val)</a>
<a href="#">nlapiGetCurrentLineItemIndex(type)</a>
<a href="#">nlapiGetCurrentLineItemMatrixValue(type, fldnam, column)</a>
<a href="#">nlapiGetCurrentLineItemText(type, fldnam)</a>
<a href="#">nlapiGetCurrentLineItemValue(type, fldnam)</a>
<a href="#">nlapiGetCurrentLineItemValues(type, fldnam)</a>

nlapiGetLineItemCount(type)
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)
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)
<b>Objects</b>
nlobjSubList 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)



<a href="#">nlapiSearchDuplicate(type, fields, id)</a>
<a href="#">nlapiSearchGlobal(keywords)</a>
<a href="#">nlapiSearchRecord(type, id, filters, columns)</a>
<b>Objects</b>
<a href="#">nlobjSearch and all methods</a>
<a href="#">nlobjSearchColumn object and all methods</a>
<a href="#">nlobjSearchFilter object and all methods</a>
<a href="#">nlobjSearchResult and all methods</a>
<a href="#">nlobjSearchResultSet and all methods</a>

## Scheduling APIs

<b>Functions</b>
<a href="#">nlapiScheduleScript(scriptId, deployId, params)</a>
<a href="#">nlapiSetRecoveryPoint()</a>
<a href="#">nlapiYieldScript()</a>

## Execution Context APIs

<b>Functions</b>
<a href="#">nlapiGetContext()</a>
<a href="#">nlapiGetDepartment()</a>
<a href="#">nlapiGetLocation()</a>
<a href="#">nlapiGetRole()</a>
<a href="#">nlapiGetSubsidiary()</a>
<a href="#">nlapi GetUser()</a>
<a href="#">nlapiLogExecution(type, title, details)</a>
<b>Objects</b>
<a href="#">nlobjContext object and all methods</a>

## UI Builder APIs

<b>Functions</b>
<a href="#">nlapiCreateAssistant(title, hideHeader)</a>



<a href="#">nlapiCreateForm(title, hideNavbar)</a>
<a href="#">nlapiCreateList(title, hideNavbar)</a>
<a href="#">nlapiCreateTemplateRenderer()</a>
<b>Objects</b>
<a href="#">nlobjAssistant object and all methods</a>
<a href="#">nlobjAssistantStep object and all methods</a>
<a href="#">nlobjButton object and all methods</a>
<a href="#">nlobjColumn object and all methods</a>
<a href="#">nlobjField object and all methods</a>
<a href="#">nlobjFieldGroup object and all methods</a>
<a href="#">nlobjForm object and all methods</a>
<a href="#">nlobjList object and all methods</a>
<a href="#">nlobjPortlet object and all methods</a>
<a href="#">nlobjSubList object and all methods</a>
<a href="#">nlobjTab object and all methods</a>
<a href="#">nlobjTemplateRenderer object and all methods</a>

## Application Navigation APIs

<b>Functions</b>
<a href="#">nlapiRequestURL(url, postdata, headers, callback, httpMethod)</a>
<a href="#">nlapiRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)</a>
<a href="#">nlapiResolveURL(type, identifier, id, displayMode)</a>
<a href="#">nlapiSetRedirectURL(type, identifier, id, editmode, parameters)</a>
<b>Objects</b>
<a href="#">nlobjRequest object and all methods</a>
<a href="#">nlobjResponse object and all methods</a>

## Date APIs

<b>Functions</b>
<a href="#">nlapiAddDays(d, days)</a>
<a href="#">nlapiAddMonths(d, months)</a>
<a href="#">nlapiDateToString(d, format)</a>
<a href="#">nlapiStringToDate(str, format)</a>



## 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
<a href="#">nlapiCreateFile(name, type, contents)</a>
<a href="#">nlapiDeleteFile(id)</a>
<a href="#">nlapiLoadFile(id)</a>
<a href="#">nlapiSubmitFile(file)</a>
Objects
<a href="#">nlobjFile object and all methods</a>

## Error Handling APIs

Functions
<a href="#">nlapiCreateError(code, details, suppressNotification)</a>
Objects
<a href="#">nlobjError object and all methods</a>

## Communication APIs

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

## Configuration APIs

Functions
<a href="#">nlapiLoadConfiguration(type)</a>
<a href="#">nlapiSubmitConfiguration(name)</a>
Objects
<a href="#">nlobjConfiguration object and all methods</a>



## SuiteFlow APIs

Functions
<code>nlapiInitiateWorkflow(recordtype, id, workflowid, initialvalues)</code>
<code>nlapiInitiateWorkflowAsync(recordType, id, workflowId, initialValueS)</code>
<code>nlapiTriggerWorkflow(recordtype, id, workflowid, actionid, stateid)</code>

## Portlet APIs

Functions
<code>nlapiRefreshPortlet()</code>
<code>nlapiResizePortlet()</code>

## SuiteAnalytics APIs

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

## User Credentials APIs

Function
<code>nlapiGetLogin()</code>
Objects
<code>nlobjLogin</code> object and all methods

## Job Manager APIs

Function
<a href="#">nlapiGetJobManager(jobType)</a>
Objects
<a href="#">nlobjJobManager object and all methods</a>
<a href="#">nlobjDuplicateJobRequest and all methods</a>
<a href="#">nlobjFuture and all methods</a>

## Record APIs

For an overview of NetSuite records, see the help topic [Working with Records in SuiteScript](#). 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\(id, baseType, baselId, altType, altId, fields\)](#)
- [nlapiMergeTemplate\(id, baseType, baselId, altType, altId, fields\)](#)
- [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 [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 [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](#)

## 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 [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 [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 [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 [Record Initialization Defaults](#) in the NetSuite Help Center.

## 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 Web site 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,TRUE,P
arent 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 mapping fileId = 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(mapping fileId);
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 mapping fileId = "CUSTIMPORTentityMultiFile";

var job = nlapiCreateCSVImport();
job.setMapping(mapping fileId);

//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.

### 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.

**i Note:** Most input validation is performed once `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 [API Governance](#) for the unit cost associated with this API.

**i Note:** Values for all required fields **must** be provided or a newly instantiated record cannot be submitted. See [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 [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 [Record Initialization Defaults](#) in the NetSuite Help Center.

## 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 [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 [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 [Record Initialization Defaults](#) in the NetSuite Help Center.

### 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 [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

[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 [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 [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, lotnumberedassemblyitem, serializedassemblyitem.
- set the type parameter to 'customer' to load the following types of entity records: customer, lead, prospect.
- set the type parameter to 'userevents', '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 [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 [Record Initialization Defaults](#) in the NetSuite Help Center.

## 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](#)

## nlapiMergeRecord(id, baseType, baseld, altType, altId, fields)

### THIS API HAS BEEN DEPRECATED

This API is deprecated as of NetSuite Version 2015 Release 1. This function will no longer be supported as of Version 2016 Release 1.



**Important:** This API only supports CRMSDK templates. CRMSDK templates are deprecated as of Version 2015 Release 1. You can convert your existing CRMSDK templates to scriptable templates within the UI.

Version 2015 Release 1 included a new set of APIs that you must use with scriptable templates. For more information on these APIs, see [nlapiCreateEmailMerger\(templateId\)](#). Note that `nlapiMergeRecord(id, baseType, baseld, altType, altId, fields)` does not support scriptable templates. If your existing scripts use `nlapiMergeRecord(id, baseType, baseld, altType, altId, fields)`, you must do the following before your NetSuite account is upgraded to Version 2016 Release 1:

- Within the UI, convert your scripted CRMSDK templates to scriptable templates. See the help topic [Converting CRMSDK Templates to Scriptable Templates](#) for additional information.
- After your templates are converted, rewrite all applicable scripts, using the new Scriptable Email Template APIs. Use [nlapiCreateEmailMerger\(templateId\)](#) as your starting point.

### Deprecated Since

- Version 2015 Release 1

## Example

The following sample is a user event script that is deployed on an `afterSubmit` event. This script applies to the Customer record type.

```
function afterSubmit(type)
{
var newRec = nlapiGetNewRecord();
var senderInfo = nlapiGetContext();
var mailRec = nlapiMergeRecord(1, 'customer', newRec.getId());
var records = new Object();
records['entity'] = newRec.getId();
nlapiSendEmail(senderInfo.getUser(), newRec.getFieldValue('email'), mailRec.getName(),
    mailRec.getValue(), null, null, records);
}
```

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

## nlapiMergeTemplate(id, baseType, baseld, altType, altId, fields)

### THIS API HAS BEEN DEPRECATED

This API is deprecated as of NetSuite Version 2008 Release 1. However, it continues to be supported. This function will not be enhanced in future versions of NetSuite.



**Important:** This API only supports CRMSDK templates. CRMSDK templates are deprecated as of Version 2015 Release 1. You can convert your existing CRMSDK templates to scriptable templates within the UI. See the help topic [Converting CRMSDK Templates to Scriptable Templates](#) for addition information.

Scriptable templates are not supported with `nlapiMergeRecord`. After your CRMSDK templates are converted, update all applicable scripts with `nlapiCreateEmailMerger(templateId)`, `nlobjEmailMerger`, and `nlobjMergeResult`. Use `nlapiCreateEmailMerger(templateId)` as your starting point.

### Deprecated Since

- Version 2008 Release 1

## Example

```
function mergeDoc(request, response)
{
    var external_fields = new Array();
    external_fields['NLCUSTOM'] = 'Custom Text'+ nlapiGetContext().getCompany();

    var document = nlapiMergeTemplate( 9, 'customer',
        request.getParameter('entity') == null ? '76' :
        request.getParameter('entity'), null, null, external_fields );
```

```

        response.setContentType('WORD')
        response.write( document );
    }

```

[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.

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 [Advanced PDF/HTML Templates \(Beta\)](#).

### 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

- PACKINGSLIP: formnumber, itemfulfillment
- PICKINGTICKET: formnumber, shipgroup, location

## 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 Web site 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

- Job ID of the import (which is also the identifier for the CSV response file)

### Since

- Version 2012.2

### Throws

This API throws errors resulting from inline validation of CSV file data before the import of data begins (the same validation that is performed between the mapping step and the save step in the Import

Assistant). Any errors that occur during the import job are recorded in the CSV response file, as they are for imports initiated through the Import Assistant.

## 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,TRUE,P
arent 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 Cab
inet

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 [nlapiCSVImport](#).

[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 [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 [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 while 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 [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 while 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 [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 [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 [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 [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 idl = nlapiSubmitRecord(trecord, true);

    nlapiSendEmail(-5, -5, 'Transform Email' + 'Original Qty = ' + qty + '' + 'Record Created =
    ' + idl , 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

Record Type	Record Name	Transform Type	Transform Name (Target Record)
prospect	Prospect	estimate	Quote
prospect	Prospect	opportunity	Opportunity
prospect	Prospect	salesorder	Sales Order
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	revenuecommitment reversal	Revenue Commitment Reversal
<p> <b>Note:</b> The return authorization must be approved and received for this transform to work.</p>			
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

Record Type	Record Name	Transform Type	Transform Name (Target Record)
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

- CANT\_VOID\_TRANS — if you attempt to void a transaction that is linked to other transactions (for example, customer payment)
- 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
Estimate	estimate
Expense Report	expensereport
Intercompany Journal Entry	intercompanyjournalentry
Invoice	invoice
Journal Entry	jurnalentry
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 <a href="#">Custom Transaction</a> .

[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 [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 [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 [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 [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 [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 [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 [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 [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 [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 [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 [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 [Using SuiteScript with / Numbered Inventory Management](#). Also see [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 [Using SuiteScript with / Numbered Inventory Management](#). Also see [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 [Using SuiteScript with / Numbered Inventory Management](#). Also see [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 [Using SuiteScript with / Numbered Inventory Management](#). Also see [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 [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**, however, 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.



**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

## Example

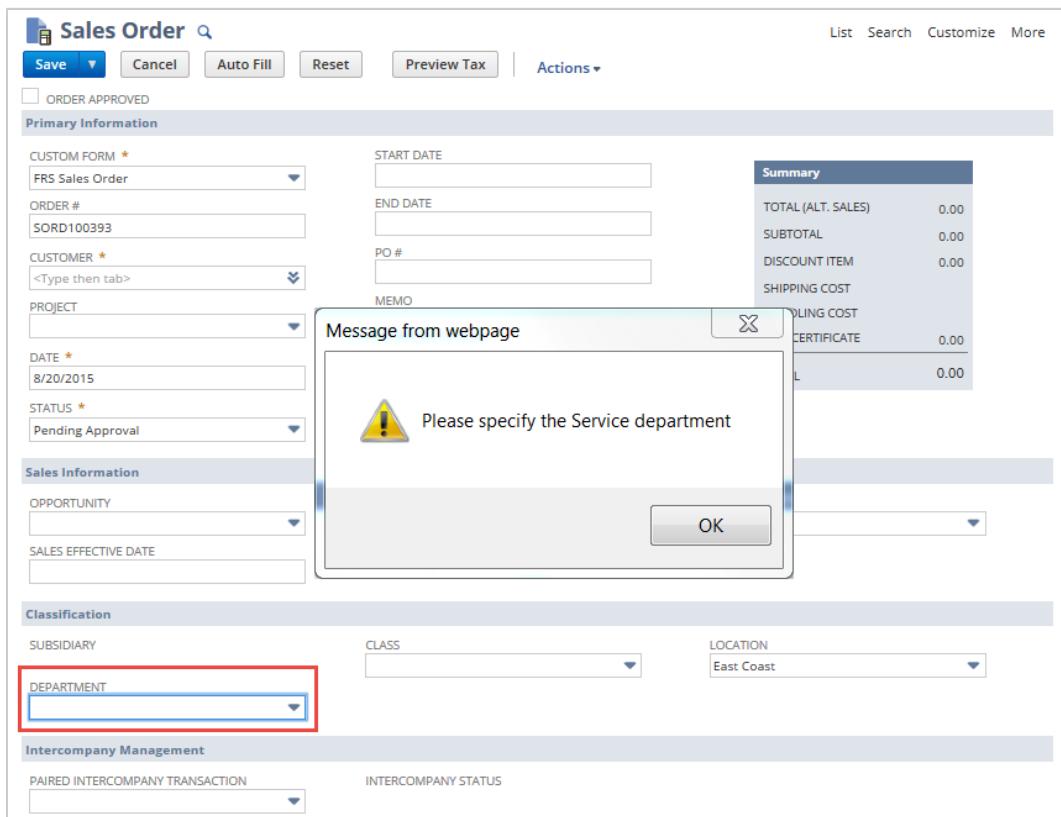
The following client script reads the text value of the Department field. If the Department field contains no value when the page loads, an alert is thrown telling users to select the Service

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');
}
}
```



**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 is 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).

### 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:** Long text fields are truncated at 4000 characters in search/lookup operations.

See [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 [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 drop-down 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()
{
nlapiSetValue('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 [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 [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 n lapiSubmitField in the context of a Suitelet.

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

//customer
nlapiSubmitField('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 [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

### Parameters



- **type {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

### Parameters

- **type {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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.

[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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 SuiteTalk.

### 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) 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 [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 [Pricing Sublist](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

## Parameters

- `type {string} [required]` - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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.

## 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) 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 [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 [Pricing Sublist](#) 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 [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 [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.

**i 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 [Pricing Sublist](#) 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 [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 [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 drop-down before adding multiple options, then enable the drop-down when finished.

## Parameters

- **type {string} [required]** - The sublist internal ID (for example, use price as the ID for the Pricing sublist). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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](#)

## nlapisLineItemChanged(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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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\(type, linenum\)](#).

**i 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) and [Demand Plan Detail Sublist](#) in the NetSuite Help Center.

### Parameters

- **type {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 pagelnit (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 [Pricing Sublist](#) 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:

The screenshot shows the 'Pricing' sublist in NetSuite. At the top, there are dropdown menus for 'QUANTITY PRICING SCHEDULE' (set to 'Pricing Schedule 1'), 'CALCULATE QUANTITY DISCOUNTS' (set to 'By Line Quantity'), and 'PRICING GROUP'. A checked checkbox 'USE MARGINAL RATES' is also visible. Below these are currency selection buttons for US Dollars, Australian Dollars, British Pounds Sterling, Canadian Dollars, Danish Krone, Euros, Japanese Yen, and Singapore Dollar. A section for 'PRICE LEVEL' and 'DEFAULT DISCOUNT %' is shown, with a red box highlighting the 'QTY 500' row which has a value of '0.0%'. To the right of this are rows for 'Default Discount %' (0.0%, 5.0%, 10.0%) and various price types like RRP, B2B Online Price, Corporate/Business/Government Customers, Shopfront Rate, and Online Price, each with their respective discount percentages.

## Parameters

- **type {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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)`

**i 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 [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 ad-hoc search, without saving it. Alternatively, calling `nlobjSearch.saveSearch(title, scriptId)` will save the search to the database, so it can be resused later in the UI or using `nlapiLoadSearch(type, id)`.

**i 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 [SuiteScript Supported Records](#).
- `filters {nlobjSearchFilter | nlobjSearchFilter[] | Object[]} [optional]` - A single `nlobjSearchFilter` object - **or** - an array of `nlobjSearchFilter` objects - **or** - a search filter expression.

**i 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_blackfriday');
```

## 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 fi
Iter
newSearch.setIsPublic(true);
newSearch.saveSearch('My new opp search', 'customsearch_blackfriday');

```

[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 [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 2012.1

## 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_blackfriday');
```

## 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 [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				
<span style="font-size: small;">+ FILTERS</span> <span style="float: right;"> </span>				
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. Results are limited to 1000 rows. Also note that in search/lookup operations, long text fields are truncated at 4,000 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 [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 [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 ad-hoc 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 `napiSearchRecord` function, see [Search Samples](#) in the NetSuite Help Center. If you are new to searching with SuiteScript, also see [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 [Scheduled Scripts](#).

## nlapiScheduleScript(scriptId, deployId, params)

A call to this API places a scheduled script into the NetSuite scheduling queue. 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 will not place the script into the scheduling queue.

If the deployment status on the Script Deployment page is set to **Scheduled**, the script will be placed into the queue according to the time(s) specified on the Script Deployment page.

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



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

One or more calls to nlapiScheduleScript can be made from Suitelet, user event, and portlet scripts. Note that you can also call nlapiScheduleScript from within a **scheduled script** to:

1. Place the currently executing scheduled script back into the scheduled script workqueue.

2. Call another scheduled script. When the new script is called, it is then put in the scheduled script workqueue.
3. Place a scheduled script into the queue from another script type, such as a user event script or a Suitelet.



**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 [Using nlapiScheduleScript to Deploy a Script into the Scheduling Queue](#).

For general details on working with scheduled scripts, see [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. If empty, the first “free” deployment will be used. Free means that the script’s deployment status appears as Not Scheduled or Completed. If there are multiple “free” scripts, the NetSuite scheduler will take the first free script that appears in the scheduling queue.



**Important:** `deployId` is a **required argument** if you are calling `nlapiScheduleScript` to requeue a scheduled script that is currently executing. This argument is also **required** if, from within a scheduled script, you are calling `nlapiScheduleScript` to queue another scheduled script that may be multiple deployments (and therefore multiple deployment IDs). In this case you must specify which of the script’s deployments you want to schedule.

- `params {Object} [optional]` - Object of name/values used in this schedule 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 [Creating Script Parameters Overview](#) in the NetSuite Help Center.

## Returns

- A string whose value is QUEUED if the script was successfully queued by this call, or it returns the script’s current status. Valid status values are:
  - **INQUEUE** - The script you requested is already in a queue 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 picked up and put into the execution queue.



**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 reschedule 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 re-queueing the current script. In the following sample, for example, nlapiScheduleScript consumes no units. Note, however, that nlapiScheduleScript is still 20 units per call if you are trying to schedule 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 the 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 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 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.
  - 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 then reschedules the script. The newly rescheduled script has its governance units reset, and is then placed at the back of the scheduled script queue. To summarize, `nlapiYieldScript` works as follows:

1. Creates a new recovery point.
2. Creates a new scheduled script with a governance reset.
3. Associates the recovery point to the scheduled script
4. Puts the script at the back of the scheduled script queue.

 **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 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.
- 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;
nlapiYieldScript();
//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\(\)](#)
- [nlapi GetUser\(\)](#)
- [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 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 Web Services Context

To cause a form to behave differently in 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 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 web services call, enable the **Disable Server-side Scripting** preference on the web services Preference page at Setup > Integration > Web Services.



**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](#)

## nlapi GetUser()

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 nlapi GetUser 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 = nlapi GetUser();

    //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 [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 [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 [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 [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 withtrandate 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 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('companynamename', 'text', 'Company Name', null, 'companyinfogroup');
assistant.addField('legalname', 'text', 'Legal Name', null, 'companyinfogroup');

assistant.addStep('entercontacts', 'Enter Contacts').setHelpText("Manually add contacts into yo
ur account.")
```



```
assistant.addStep('importdata', 'Import Data').setHelpText("Finally, import records into your account via CSV.");

response.writePage(assistant);
```

[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 Suitlets or other web sites 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](#)

## 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 1.0, TLS 1.1, and TLS 1.2. NetSuite recommends that you use TLS 1.2. For more information, see the help topics [FAQ: Transport Layer Security \(TLS\) Deprecations](#), specifically [SuiteScript and TLS](#).

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](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.

- **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

### 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://webservices.netsuite.com/wsdl/v1_2_0/netsuite.wsdl', 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://webservices.netsuite.com/wsdl/v1_2_0/netsuite.wsdl', null, a, handleResponse);
function handleResponse(response)
{
    var headers = response.getAllHeaders();
    var output = 'Code: '+response.getCode()+'\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://webservices.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.
- **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 nlapiRequestURLWithCredential s.

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');

nlapiSetValue('custentity_custompassword', credValue);
nlapiSetValue('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', 'system.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://system.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 nlPopupHelp string. For example, this search might return onclick="nlPopupHelp('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 | boolean}** [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**.



**Important:** If the type argument is set to SUITELET or RESTLET, set displayMode to true to return an external URL. Set displayMode to false, or simply omit the argument, to return an internal URL. For Suitelets and RESTlets, displayMode automatically defaults to false.

## 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.

## 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.

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
  - **datetimetz** — 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.
- **datetimetz** — 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [Olson Values](#) table.

**Returns**

- void

**Throws**

- SSS\_INVALID\_ARG\_TYPE

**Since**

- Version 2013 Release 2

**Example**

```
nlapiSelectNewLineItem('recmachcustrecord_childdatetime');
nlapiSetCurrentLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzco',
    '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 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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, lineNumber, 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [Olson Values](#) table.

## Returns

- **void**

## Throws

- SSS\_INVALID\_ARG\_TYPE

## Since

- Version 2013 Release 2

## Example

```
nlapiSetLineItemDateTimeValue('recmachcustrecord_childdatetime', 'custrecord_datetimetetzcol', 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.

- [nlapiExchangeRate\(sourceCurrency, targetCurrency, effectiveDate\)](#)
- [nlapiFormatCurrency\(str\)](#)

## [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.

Currency Exchange Rates					
		TOTAL: 5			
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. 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

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/>

## 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' ''>\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 += "</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='russianfont1' 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>`, `<b>` by `</b>` and so on. When an element has no content, like `<br>`, `<img>` 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\(id, baseType, baselId, altType, altId, fields\)](#)

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 name of the file
- type {string} [required] - The file type. For a list of supported file types, see [Supported File Types](#) in the NetSuite Help Center. Note that when specifying the type for an ad-hoc 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 while 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 recipient must have an email.

## 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_campaignndrip_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_campaignndrip_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 5MB.
- To send an email message from one address and to receive replies at another address. To do this, use the replyTo parameter.

**i** **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\(id, baseType, baseld, altType, altId, fields\)](#), used to perform mail merges with CRMSDK templates. This function will no longer be supported as of Version 2016 Release 1.

**i** **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

- **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
  - 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 `napiMergeRecord(id, baseType, baseld, altType, altId, fields)` [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)
<b>transaction</b> (use for transaction and opportunity record types)	<code>records['transaction'] = '1000';</code>
<b>activity</b> (use for Case and Campaign record types)	<code>records['activity'] = '50';</code>
<b>entity</b> (use for all Entity record types, for example, customer, contact, etc.)	<code>records['entity'] = '555';</code>
<b>record</b>	<code>records['record'] = '3';</code>

Key	Value (examples)
(custom record internalId - for custom records you must also specify both the record ID and the record type ID)	
<b>recordtype</b> (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).
- **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_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, `jwolfe@netsuite.com`, receives a bounceback notification for each recipient email not successfully delivered.

```
nlapiSendEmail('jwolfe@netsuite.com', ['msample@netsuite.com', 'jdoe@netsuite.com'],
    'hello world', 'your order has been completed',
    ['sales@netsuite.com', account-management@netsuite.com'], mySalesOrder, myPdf, t
rue);
```

### Example 5

This example shows how to send an email from the original sender, `jwolfe@netsuite.com`, to a recipient, `customer@customer.com`. The reply-to field of the email will be set to `accounts@netsuite.com`.

```
nlapiSendEmail('jwolfe@netsuite.com', '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 NetSuite's 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 to use SuiteBundler to deploy your scripts into other accounts.

The screenshot shows the 'SuiteSignOn' setup page. It includes fields for NAME (SSO Portlet), ID (containing '\_wif\_sso\_partner\_portlet'), CONSUMER KEY (sZmB7DxHBZJwPlc), SHARED SECRET, CONFIRM SHARED SECRET, PARTNER ACCOUNT, WEB SERVICES ACCESS (set to 'Same As UI Role'), and an INACTIVE checkbox. The 'ID' field is highlighted with a red box.

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 **customss**.

To see a list of IDs for all SuiteSignOn records, go to the SuiteSignOn list page (Setup > Integration > SuiteSignOn).

### Returns

- URL, OAuth token, and any integration variables as a string

### Throws

- SSS\_SUITESSIGNON\_NOT\_CONFIGURED
- SSS\_INVALID\_SUITESSIGNON

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('customssso_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', 'SSO1');
label.setDefaultValue ('<B>Check out my SSO Suitelet!!</B>');

var url = nlapiOutboundSSO('customssso_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', 'SSO2');
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('customssso_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 [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 [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 workflow base record (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 of the workflow base record (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 workflow base record (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](#)

## nlapiCreateReportForm(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 [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
var login = nlapiGetLogin();
```

[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 NetSuite's 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 NetSuite's 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 when 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 DUPLICATEREORDS.

### 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('DUPLICATEREORDS');

    // 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 is an array of duplicate record IDs
mergeJobRequest.setRecords(duplicateRecords);

// 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 [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
// check box 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 [nlapiGetContext\(\)](#) function returns a reference to this object.

### nlobjContext Methods

- [getBundleId\(\)](#)
- [getColorPreferences\(\)](#) (Deprecated as of Version 2014 Release 2)
- [getCompany\(\)](#)
- [getDepartment\(\)](#)
- [getDeploymentId\(\)](#)
- [getEmail\(\)](#)
- [getEnvironment\(\)](#)
- [getExecutionContext\(\)](#)
- [getFeature\(name\)](#)
- [getLocation\(\)](#)
- [getLogLevel\(\)](#)
- [getName\(\)](#)
- [getPercentComplete\(\)](#)
- [getPermission\(name\)](#)
- [getPreference\(name\)](#)
- [getQueueCount\(\)](#)



- `getRemainingUsage()`
- `getRole()`
- `getRoleCenter()`
- `getRoleId()`
- `getScriptId()`
- `getSessionObject(name)`
- `getSetting(type, name)`
- `getSubsidiary()`
- `getUser()`
- `getVersion()`
- `setPercentComplete(pct)`
- `setSessionObject(name, value)`
- `setSetting(type, name, value)`

## getBundleId()

Returns the bundle ID for the current script.

Returns

A string value representing the bundle ID for 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
- **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 [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

**i 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 > Status.

```
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 [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](#).

**i 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 [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](#)

## getQueueCount()

Returns the number of scheduled script queues in a specific account.

This method is helpful for SuiteApp developers who want to check for the number of queues in an account. If the consumer of the SuiteApp has purchased a SuiteCloud Plus license, a call to `getQueueCount` will return 5, meaning that the account has 5 scheduled script queues. A call to `getQueueCount` in accounts that do not have a SuiteCloud Plus licence will return 1, meaning the account has only 1 scheduled script queue. (Note that in some cases, an account may have two licenses supporting 10 queues, or three licenses supporting 15 queues.)

When you get the number back in script, you can make business logic decisions based on that number. For example, if you know an account has 5 queues, you can have more than 1 script deployment and distribute the processing load to more than 1 queue.

## 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.
  - **FEATURE** - returns T (enabled) or F (disabled) depending on whether a feature is enabled. Supported in client and server SuiteScript.



**Important:** The SESSION type value is not supported in Client SuiteScript.

In the NetSuite Help Center, see [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 [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 [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 [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 > Status. See [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 Re
    p
    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 a string 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(string)`
- `base64()`
- `md5()`
- `replace(string1, string2)`
- `sha1()`
- `sha256()`
- `utf8()`

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

## append(string)

Appends a passed in string to an `nlobjCredentialBuilder` object.

### Parameter

- `string {string}` [required] — string 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 instances 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, once for each linked file.

## Parameters

- **sublist {string} [required]** — The internal ID of the record sublist for which data is being imported. See [Scriptable Sublists](#) for a list of sublist internal IDs.
- **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 `<nlobjDuplicateJobRequest>.<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 '+nlapiGetRecordType()+' 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 Technical 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 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 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\(id, baseType, baseld, altType, altId, fields\)](#)
- [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 [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.

**i 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 nlobjRequest.getFile method can return a reference to a file up to 10MB in size.

 **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
        nlapiAttachRecord("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**.

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 [nlobjJobManager.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 return 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 `nlobjFuture` 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('MycUrr3ntPa$$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('MycUrr3ntPa$$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.

**i 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.

## Methods

- [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\(group, fldnam, lineum, column\)](#)
- [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, date, timeZone)`
- `setCurrentLineItemMatrixValue(group, fldnam, column, value)`
- `setCurrentLineItemValue(group, name, value)`
- `setDateTimeValue(fieldId, date, timeZone)`
- `setFieldText(name, text)`
- `setFieldTexts(name, text)`
- `setFieldValue(name, value)`
- `setFieldValues(name, value)`
- `setLineItemDateTimeValue(type, fieldId, lineNum, date, 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)`
- `nlapiTransformRecord(type, id, transformType, transformValues)`

## 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.

- 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 [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 [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 [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 [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 [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 [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 [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 [Pricing Sublist](#) in the NetSuite Help Center.

## Parameters

- **group {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.

## 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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.getCurrentLineItemDateValue('recmachcustrecord_childdatetime', 'custrecord_datedatetimetzcol', '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 [Pricing Sublist](#) in the NetSuite Help Center.

## Parameters

- group {string} [required] - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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 custen
tity1 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.

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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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.getLineItemDate TValue('recmachcustrecord_childdatetime', 'custrecord_datetimetzco
!', 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- 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 [Pricing Sublist](#) in the NetSuite Help Center.

### Parameters

- group {string} [required] - The sublist internal ID. In the NetSuite Help Center, see [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 [Pricing Sublist](#) in the NetSuite Help Center.

## Parameters

- **group** {string} [required] - The sublist internal ID. In the NetSuite Help Center, see [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

```
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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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 `getLineItemText(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.

**i 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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 [Pricing Sublist](#) 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 [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 [Pricing Sublist](#) 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 [Matrix APIs](#) in the NetSuite Help Center.

## Parameters

- **group {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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('price1', '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 [Pricing Sublist](#) 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 [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 [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- 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 [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 [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 [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.



## 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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.setCurrentLineItemDateValue('recmachcustrecord_childdatetime', 'custrecord_datetimetetzcol'
, '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 [Pricing Sublist](#) in the NetSuite Help Center.

## Parameters

- **group {string} [required]** - The sublist internal ID. In the NetSuite Help Center, see [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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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, date, time, 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.
- **date {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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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, lineNumber, 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 [Olson Values](#) table (values are case-insensitive). If an integer is passed in, it must match one of the Key values listed in the [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, '0
1/01/2013 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- **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 [Pricing Sublist](#) 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 [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 [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 [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 [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 [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 [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':colum
```

```
ns[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 `napiCreateReportForm`.

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 = napiCreateReportForm('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', nul
    l, '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	
		Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %	Projected Total	Probability %
Aaron Rosewall-Godley	Mathew Christer	\$3,692.00									
	Mathew Christer			\$26,644.20							
Total - Aaron Rosewall-Godley		\$3,692.00		\$26,644.20		\$0.00		\$0.00		\$0.00	
Abdullah Bhupathiraju	Luke Duke	\$3,692.00									
Adam Fitzpatrick	Jon Baker					\$5,000.00					
Adam Kavelmacher	J Wolfe	\$2,300.00									
Adelina Shanikwiler	Mathew Christer	\$14,849.66									
Adina Fitzpatrick	J Wolfe			\$4,300.00							
Andrew Goldwasser	Jon Baker						\$14,478.15				
Apu Nahasapeemapetilon	Jon Baker	\$18,861.80									
Barney Brooking	Jon Baker						\$8,544.00				
Barney Gumble	Jessie Barto						\$3,692.00				
Beverly Linden	Theodore Hosch						\$8,532.75				
Billing Dental Clinic	Christian Begum							\$7,384.00			
	Theodore Hosch							\$4,293.90			
	Theodore Hosch	\$6,330.75									
	Theodore Hosch	\$4,660.88									
Total - Billing Dental Clinic	J Wolfe								\$5,100.00		
		\$10,991.63		\$0.00		\$0.00		\$11,677.90		\$5,100.00	
											\$27,769.53

You can use the UI to define the row/column hierarchy and the actual column data of a pivot report. For more information, see the help topic [Creating a Pivot Report](#).

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\(name\)](#)
- [getURL\(\)](#)

## getAllHeaders()

Returns an Object containing all the request headers and their values.



## Returns

- `String[]` of header names

## 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). The getFile method can return a reference to a file up to, but not including, 10MB in size.

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 header in the request

Parameters

- name {string} [required]- The name of the header

Returns

- The request header 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.

## 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.

**i 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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of sublists that support SuiteScript, as well as all internal IDs associated with each sublist.
- 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](#)

## getURL()

Returns the full URL of the request

### 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 [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\(type\)](#)
- [setContent-Type\(type, name, disposition\)](#)
- [setEncoding\(encodingType\)](#)
- [setHeader\(name, value\)](#)
- [sendRedirect\(type, identifier, id, editmode, parameters\)](#)
- [write\(output\)](#)
- [writeln\(output\)](#)
- [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 `nlapiRequestURL`.

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 `nlapiXMLToPDF(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" "repor
t-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 [Supported File Types](#) in the NetSuite Help Center.
- **name {string} [optional]** - Set the name of the file being downloaded (for example 'foobar.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.

### 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 [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');
// CalladdColumn 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](#)

## addColumns(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.addColumn(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.

**i 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.

**i 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.

**i 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 ad-hoc 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 2012.1

### 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 ad-hoc 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 ad-hoc 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 ad-hoc 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.

**i 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 [nlapiCreateSearch\(type, filters, columns\)](#).

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 [nlapiCreateSearch\(type, filters, columns\)](#).

### 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 [nlapiLoadSearch\(type, id\)](#) 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 = nlapiCreateSearch( '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 ad-hoc 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 ad-hoc 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 [FSuiteScript 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 [Searching Overview](#) in the NetSuite Help Center.

**i 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 [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

Figure 3 shows the values for **label** and **formula** for the **Customer Names (Reverse)** column.

## Figure 3. Figure 3

```

value = {string} Lincoln, Abe
functionName = null
formula = {string} {lastname} || ', ' || {firstname}
label = {string} Customer Names (Reverse)
column = {nlobjSearchColumn} formulatext
  name = {string} formulatext
  join = null
  summary = null
  label = {string} Customer Names (Reverse)
  function = null
  formula = {string} {lastname} || ', ' || {firstname}
  sortdir = null
  type = {string} text

```

Figure 4 shows the values for **label** and **functionName** for the **Customer Balance** column.

## Figure 4. Figure 4

```

value = {string} Lincoln, Abe
functionName = {string} round
formula = null
label = {string} Customer Balance
column = {nlobjSearchColumn} balance
  name = {string} balance
  join = null
  summary = null
  label = {string} Customer Balance
  function = {string} round
  formula = null
  sortdir = null
  type = {string} currency
  joinlabel = null
  searchtype = null

```

[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 [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



ID	Name	Date Function	Output
negate	Negate	No	
numberAsTime	Number as Time	No	text
quarter	Quarter	Yes	text
rank	Rank	No	integer
round	Round	No	
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 [Searching Overview](#) in the NetSuite Help Center.

**i 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 [Search Operators](#).

**i 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 [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 [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_a1');
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 drop-down 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 [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 [Canceling an Inventory Detail Subrecord](#).

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

### commit()

Use this method to commit the subrecord to the parent record. See [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 [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 [Creating Custom NetSuite Pages with UI Objects](#) for more information.
- While 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 [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 [Create a Form with a URL Field](#) for an example how to use this type.
  - `timeofday`
  - `textarea`
  - `multiselect`
  - `image`
  - `inlinehtml`
  - `password`
  - `help`



- percent
- longtext
- 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 appear 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

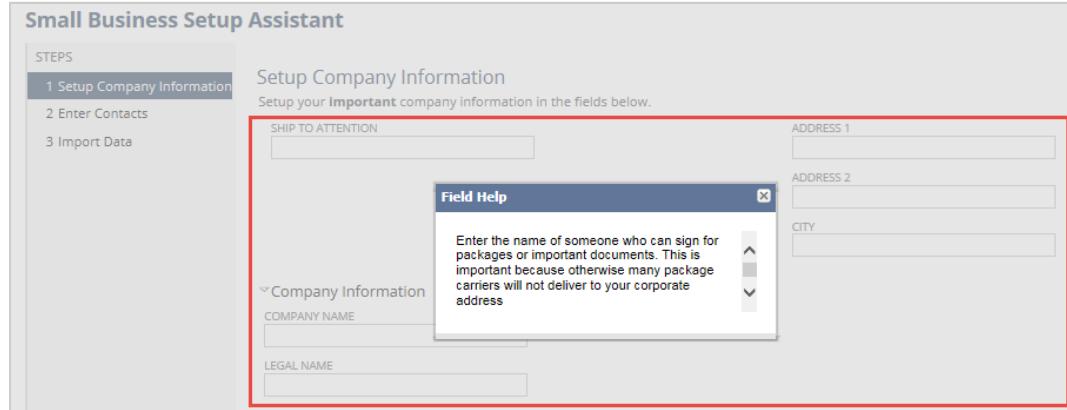
- Version 2009.2

### 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.");
```

The screenshot shows a SuiteScript assistant titled "Small Business Setup Assistant". On the left, a sidebar lists three steps: "1 Setup Company Information" (highlighted in blue), "2 Enter Contacts", and "3 Import Data". The main content area is titled "Setup Company Information" and contains the following text in a red box: "Setup your **important** company information in the fields below.". Below this text are several input fields: "SHIP TO ATTENTION", "ADDRESS 1", "ADDRESS 2", and "CITY". At the bottom of the form are buttons for "Cancel", "< Back", and "Next >".

[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**

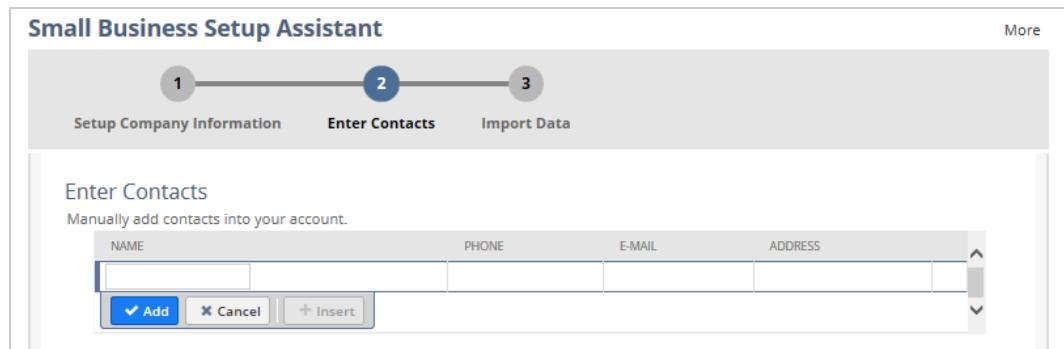
## Since

- 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 [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 [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 [nlapiCreateAssistant\(title, hideHeader\)](#) 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 [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 [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 [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 [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:

```
nlapiSetRedirectURL('suitelet', nlapiGetContext().getScriptId(), nlapiGetContext().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 [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 [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.");
```

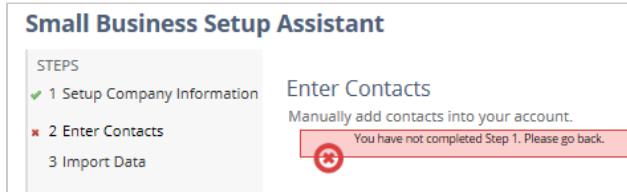
```

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, U
SA"});

```

[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 [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**:



**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.

SHIP TO ATTENTION

ADDRESS 1   
ADDRESS 2   
CITY

Company Information

COMPANY NAME   
LEGAL NAME

ordered parameter set to false :

**Small Business Setup Assistant**

1      2      3

Setup Company Information    Enter Contacts    Import Data

Setup Company Information  
Setup your **Important** company information in the fields below.

SHIP TO ATTENTION

ADDRESS 1   
ADDRESS 2   
CITY

Company Information

COMPANY NAME   
LEGAL NAME

## 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]** - ThescriptId 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 [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 [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 [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 [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 [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.
- 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
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

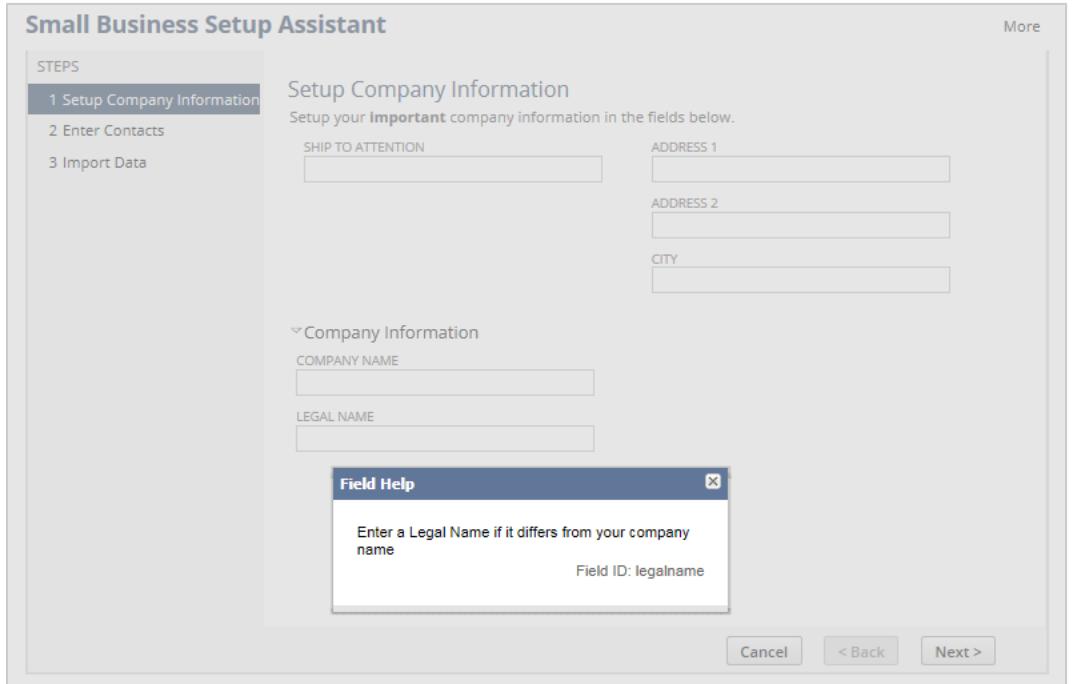
- Version 2009.2

### 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);
```



[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 [Setting 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 [Setting 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 [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.

The screenshot shows the 'Small Business Setup Assistant' interface. On the left, a sidebar lists 'STEPS': 1 Setup Company Information (selected), 2 Enter Contacts, and 3 Import Data. The main area is titled 'Setup Company Information' with the sub-section 'Company Preferences'. This section contains fields for 'FIRST DAY OF THE WEEK' (a dropdown menu), a checked checkbox for 'USE STATE ABBREVIATIONS IN ADDRESSES', and a text input for 'CUSTOMER CENTER WELCOME MESSAGE'. Below this is a collapsed section 'Accounting Preferences' and another collapsed section 'More Accounting Preferences' which includes a checkbox for 'USE ACCOUNT NUMBERS' and a dropdown for 'DEFAULT EXPENSE ACCOUNT'. At the bottom are 'Cancel', '< Back', and 'Next >' buttons.

[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)`.



## 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).

## 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 `nlobjRequestUrlWithCredentials`. Note that although no exception is thrown if this parameter value is not passed, `nlobjRequestURLWithCredentials(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, `nlobjRequestURLWithCredentials(credentials, url, postdata, headers, httpsMethod)` will not work without it.
- **value {string} [optional]** - If you choose, you can set an initial value for this field. This value is the handle to the credentials.
- **entityMatch {boolean} [optional]** - Controls whether use of `nlobjRequestUrlWithCredentials` 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 `nlobjRequestURLWithCredentials(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 = nlobjCreateForm('Simple Form');
        form.addCredentialField('username', 'Username', 'www.mysite.com', 'customscript_a', null,
false, null);
        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`
  - `datetimetetz` - 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 [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
- richtext
- file - This field type is available only for Suitelets and will appear on the main tab of the Suitelet page. 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.

## 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_academic
historytab');

//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_studentcont
acttab');
```



## 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

Record Type	Internal ID	Internal ID (number)
Customer Category	customercategory	-109
Department	department	-102
Email Template	emailtemplate	-120
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, while 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').setDefaultValue( '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?
<input type="radio"/> BUSINESS TO CONSUMER	<input type="radio"/> SMALL (0-99 EMPLOYEES)
<input checked="" type="radio"/> BUSINESS TO BUSINESS	<input type="radio"/> MEDIUM (100-999 EMPLOYEES)
<input type="radio"/> NON-PROFIT	<input type="radio"/> LARGE (1000+ EMPLOYEES)

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').setDefaultValue( '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);
}

```

**Simple Form**

[More](#)

[Submit](#)

**My Field Group**

COMPANY NAME <input type="text"/>	TEXTAREA <input type="text"/>
LEGAL NAME <input type="text"/>	
DATE <input type="text"/>	
CURRENCY <input type="text"/>	

**Academic History Student Contact**

[University Information](#)

UNIVERSITY NAME <input type="text"/>	
UNIVERSITY ADDRESS <input type="text"/>	

[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_academic
historytab');

//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_studentcont
acttab');
```



[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 `napicreateForm`, 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 = napicreateForm('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_academic
historytab');

//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_studentcont
acttab');

//Get all tabs in the Interns form.
var tabs = newForm.getTabs();
```

[Standard Objects](#) | [UI Objects](#) | [SuiteScript Functions](#)

## insertField(field, nextId)

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 [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 [napiCreateList\(title, hideNavbar\)](#) 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



## 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 [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 [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 [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 [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 [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 [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 [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 [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 [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 [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)

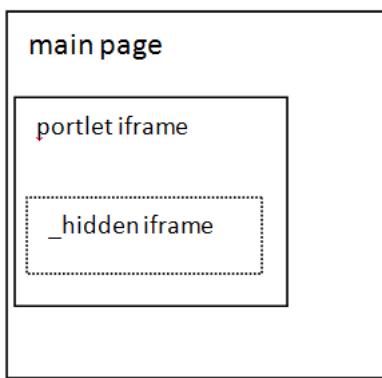
Adds a SUBMIT button with an optional custom label to this FORM portlet.

This API is only available if the portlet type is a **FORM** type. (In the NetSuite Help Center, see [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_su
itelet'), 'Save', '_hidden');
  
```

**i 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 shows the NetSuite home page. At the top, there's a navigation bar with links like Activities, Payments, Transactions, Lists, Reports, Customization, Documents, Setup, Training, and Support. Below the navigation bar, there's a "Home" section. On the left, there's a "Flash Portlet" containing an apple icon and the text "Christy's Catering". To its right is a "Simple Form Portlet" (highlighted with a red box) containing fields for TEXT, INTEGER, DATE, SELECT (with options Oranges and Apples), and a TEXTAREA. Below these fields is a "Submit" button. To the right of the simple form is a "Estimates List Detail" table with 13 rows of data. At the bottom of the page is a "Calendar: My Calendar" section 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 `nlobjSublist`.

### 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`
  - `datetimetetz` - 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.addButton("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). In the NetSuite Help Center, see [Scriptable Sublists](#) for a list of 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()`

## 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](#)



# SuiteScript 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 API Overview](#).

## Functions

- [nlapiAddDays\(d, days\)](#)
- [nlapiAddMonths\(d, months\)](#)
- [nlapiAttachRecord\(type, id, type2, id2, attributes\)](#)
- [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\(\)](#)
- [nlapi GetUser\(\)](#)
- [nlapiInitiateWorkflow\(recordtype, id, workflowid, initialvalues\)](#)
- [nlapiInsertLineItem\(type, line\)](#)
- [nlapiInsertLineItemOption\(type, fldnam, value, text, selected\)](#)



- `nlapiInsertSelectOption(fldnam, value, text, selected)`
- `nlapisLineItemChanged(type)`
- `nlapiLoadFile(id)`
- `nlapiLoadRecord(type, id, initializeValues)`
- `nlapiLoadSearch(type, id)`
- `nlapiLogExecution(type, title, details)`
- `nlapiLookupField(type, id, fields, text)`
- `nlapiMergeRecord(id, baseType, baseld, altType, altId, fields)`
- `nlapiMergeTemplate(id, baseType, baseld, 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](#)

# SuiteScript Reference

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following topics are covered in this section. If you are new to SuiteScript, these topics should be read in order.

- [How to Use SuiteScript Records Help](#)
- [SuiteScript References Overview](#)
- [Working with the SuiteScript Records Browser](#)

## How to Use SuiteScript Records Help

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The SuiteScript reference guide must be used in conjunction with the SuiteScript Records Browser. Each record listed in this guide includes a link to the Records Browser. Within the Records Browser, you will find all available fields, sublists, and search filter fields for that record. The Records Browser also provides field level help for many fields that appear on the record. Within this guide, the description of each record often includes the following types of information:

- **Supported script types** — a description of whether the record is supported in both client and server SuiteScript, and whether the user events are supported.
- **Supported functions** — a list of all functions that can be used with the record.
- **Field definitions** — provides a link to the Records Browser and, in some cases, additional explanations about how to work with particular fields.
- **Usage notes** — additional details or context about working with the record.
- **Code samples** — examples of how to structure your scripts.

 **Important:** For a list of all NetSuite records that are supported in SuiteTalk, see [SuiteScript Supported Records](#).

## Working Online

If you are working online, using either the PDF or online help version of this guide, click the links to the SuiteScript Records Browser that are provided with each record description. The Records Browser will open in your default browser.

## Downloading the SuiteScript Records Browser

If you want to use the SuiteScript Records Browser while working offline, you must download the [SuiteScript Records Browser .zip file](#)

After downloading the .zip file to your local machine, extract the .zip and navigate to the script directory. To view the Records Browser content, open index.html file in the browser of your choice.



**Note:** For information on using the Records Browser, see [Working with the SuiteScript Records Browser](#).

## SuiteScript References Overview



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The NetSuite Help Center provides all the internal IDs you will need when referencing NetSuite records, fields, sublists, search filters, permissions, etc., in SuiteScript.



**Important:** When writing SuiteScript, you must use the internal IDs listed in the NetSuite Help Center. Although you can access internal IDs by viewing page source on a NetSuite record, there is no guarantee that all IDs in the source code are supported in SuiteScript. If you create a script that references an unsupported/undocumented ID, and the ID is later changed by NetSuite, your script may break.

To find SuiteScript-supported records and internal IDs:

1. Open the [SuiteScript Records Browser](#). Only records that officially support SuiteScript are listed in the SuiteScript Records Browser. (For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).)
2. Click the record you want to reference in SuiteScript.

You will see all internal IDs currently supported for this record. These IDs can include:

- field IDs - used when referencing [Field APIs](#)
- sublist internal IDs - used when referencing [Sublist APIs](#)
- sublist field internal IDs - used when referencing [Sublist APIs](#)
- search join IDs - used when referencing [Search APIs](#)
- search filter IDs - used when referencing [Search APIs](#)
- search column IDs - used when referencing [Search APIs](#)
- transformation IDs - used when working with `nlapiTransformRecord(type, id, transformType, transformValues)`



**Important:** Not every field that appears in the [SuiteScript Records Browser](#) is settable through SuiteScript. Some fields are read-only. You will need to look at the NetSuite UI to know whether a field is settable. The general rule is that if you can set a field in the UI, you can set it in SuiteScript. If you cannot set a field in the UI, you cannot set it using SuiteScript. You can, however, still get the field's value using SuiteScript.

In the NetSuite Help Center, also see these links for additional internal ID information. The IDs listed in these sections do not pertain to specific NetSuite records.

# Working with the SuiteScript Records Browser



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The [SuiteScript Records Browser](#) provides a summary of all records, fields, sublists, search joins, search filters, search columns, and record transformations that are supported in SuiteScript. Information about elements is displayed as a series of tables. For more details, see the following topics:

- [Finding a Record or Subrecord](#)
- [Understanding the Record Summary](#)



**Note:** You can use the Records Browser online, or you can download it. For help downloading the browser, see [Downloading the SuiteScript Records Browser](#).



**Note:** For information on governance applied to individual SuiteScript APIs, as well as to various SuiteScript script types, see [SuiteScript Governance](#).



**Note:** Only the Records Browser for the most recent release of NetSuite is supported. Older version may still be accessible, but they are not supported.

## Finding a Record or Subrecord



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

To find a record or subrecord in the [SuiteScript Records Browser](#), use the A-Z index at the top of the browser window.

### To find a record or subrecord:

1. Click the appropriate letter at the top of the browser window.



The pane at the left updates to include a list of all records whose names begin with the letter you selected. The center pane updates to show details of the first record in the list.

2. In the left-hand pane, click the name of the record you are interested in.  
The center pane updates to show details of the record.

## Understanding the Record Summary

In the [SuiteScript Records Browser](#), for each record exposed to SuiteScript, you can find a reference page, such as the one in the following illustration.

The screenshot shows the SuiteScript Records Browser interface. At the top, there are three tabs: 'Schema Browser' (disabled), 'Records Browser' (selected and highlighted in yellow), and 'Connect Browser'. Below the tabs is a navigation bar with letters A through W. The 'Customer' record is selected, indicated by a yellow background in the left sidebar. The main content area displays the record's internal ID ('customer'), fields, and a table of its fields.

Internal ID	Type	nlapiSubmitField	Label
accessrole	select	false	Role
accountnumber	text	true	Account
altemail	email	true	Alt. E-mail
altphone	phone	true	Alt. Phone
autoname	checkbox	false	Auto
balance	currency	false	Balance
billaddr1	text	false	
billaddr2	text	false	
billaddr3	text	false	
billcity	text	false	
billcountry	text	false	
billpay	checkbox	false	Enable Online Bill Pay
billstate	text	false	

## Summary of the Record

For each record, the browser displays a series of tables summarizing the following:

- Fields — The record's fields.
- Sublists — A table representing each supported sublist.
- Tabs — A list of tabs available on the record in the user interface.
- Search Joins — A list of other searches you can access while searching this record.
- Search Filters — Fields in the record that you can use as search criteria.
- Search Columns — Fields that you can include search results.
- Transform Types — A list of records that this record can be transformed into using `nlapiTransformRecord`.

Some of the column names used by these tables are described below.

Column Label	Description
Name	The internal ID of the field.
Type	The data type of the field.
nlapiSubmitField	A boolean value indicating whether the field supports the <code>nlapiSubmitField</code> function, which enables you to do the equivalent of direct line editing. For details on this function, see <a href="#">Inline Editing Using nlapiSubmitField</a> .
Label	The label for the field as shown in the user interface.
Help	Additional details about working with the field.

## Comparing SuiteScript, SuiteTalk, and SuiteAnalytics Connect Exposure

To check whether the record you are currently viewing is also supported in SuiteTalk or SuiteAnalytics Connect, click the Schema Browser tab or the Connect Browser tab at the top of the page.



If the record is supported in SuiteTalk or SuiteAnalytics Connect, you are directed to the corresponding page in the SuiteTalk Schema Browser or SuiteAnalytics Connect Browser. Otherwise, you are directed to the first page of the respective browser. To compare record types support across SuiteScript, SuiteTalk, and SuiteAnalytics Connect, see the help topic [SuiteCloud Supported Records](#).

# SuiteScript Supported Records



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists all NetSuite records that support SuiteScript. Also provided are record IDs, which are often referenced in SuiteScript APIs.

Note that a scripting level defined as **Full** means that the record can be created, updated, copied, deleted, and searched using SuiteScript.

All subrecords are scriptable from the line item level, unless stated otherwise.

Memorized transactions do not support SuiteScript.

Record Name	Record ID	Record Category	Scripting Level	Scriptable in Client SuiteScript	Scriptable in Server SuiteScript
Account	account	List	Full	Record-level Client SuiteScript only	X
Accounting Book	accountingbook	List	Copy Not Allowed		X
Accounting Context	accountingcontext	List	Read, Create, Edit, Copy, Delete, and Search	X	X
Accounting Period	accountingperiod	List	Read and Search Only	X	X
Address See Using SuiteScript with Address Subrecords.	addressbookaddress	Subrecord	See Using SuiteScript with Address Subrecords.	X	Server-side scripts must access through the parent record.
Allocation Schedule	allocationschedule	List	Create, Read, Update, and Delete	X	X
Amortization Schedule	amortizationschedule	List	Copy, Create, and Delete Not Allowed		X
Amortization Template	amortizationtemplate	List	Full		X
Activity	activity	Activity	Search Only	X	X
Assembly Build	assemblybuild	Transaction	Full		X
Assembly Item	assemblyitem	Item	Full		X
Assembly Unbuild	assemblyunbuild	Transaction	Full		X
Billing Class	billingclass	List	Full		X
Billing Rate Card	billingratecard	List	Read, Create, Edit, Delete, and Search Copy and transform not supported	X	X
Billing Schedule	billingschedule	List	Full		X
Bin	bin	List	Full		X
Bin Putaway Worksheet	binworksheet	Transaction	Copy and Update Not Allowed		Dynamic Mode Only
Bin Transfer	bintransfer	Transaction	Full		X
Blanket Purchase Order	blanketpurchaseorder	Transaction	Full	X	X
Campaign	campaign	Marketing	Full	X	X
Campaign Template	campaigntemplate	Marketing	Search Not Allowed		X
Case	supportcase	Support	Full	X	X
Cash Refund	cashrefund	Transaction	Full	X	X
Cash Sale	cashsale	Transaction	Full	X	X
Charge	charge	Transaction	Full		X
Check	check	Transaction	Full	X	X
Class	classification	List	Full	X	X

Record Name	Record ID	Record Category	Scripting Level	Scriptable in Client SuiteScript	Scriptable in Server SuiteScript
Commerce Category	commercecategory	Website	Attach, Transform, and Copy are not supported		X
Competitor	competitor	Entity	Full	X	X
Consolidated Exchange Rate	consolidatedexchange rate	List	Read, Update, and Search only Create and Delete are not supported	X	X
Contact	contact	Entity	Full	X	X
Coupon Code	couponcode	Marketing	Full	X	X
Credit Card Charge	creditcardcharge	Transaction	Full		X
Credit Card Refund	creditcardrefund	Transaction	Full		X
Credit Memo	creditmemo	Transaction	Full	X	X
Currency	currency	List	Full — with Multiple Currencies feature enabled Read Only — without Multiple Currencies feature		X
Customer	customer	Entity	Full	X	X
Customer Category	customercategory	List	Search Not Available	X	X
Customer Deposit	customerdeposit	Transaction	Full	X	X
Customer Payment	customerpayment	Transaction	Copy and Create Not Allowed		X
Customer Refund	customerrefund	Transaction	Full		X
Custom List	customlist *	Custom	Search Only	X	X
Department	department	List	Full	X	X
Deposit	deposit	Transaction	Copy Not Allowed Dynamic mode is not supported	X	X
Deposit Application	depositapplication	Transaction	Create Not Allowed	X	X
Description	descriptionitem	Item	Full	X	X
Discount	discountitem	Item	Full	X	X
Download Item	downloaditem	Item	Full	X	X
Email Template	emailtemplate	Marketing	Search Not Allowed		X
Employee	employee	Entity	Full	X	X
Entity	entity	Entity	Search Only		
Estimate / Quote	estimate	Transaction	Full	X	X
Event	calendarevent	Activity	Full	X	X
Expense Category	expensecategory	List	Full		X
Expense Report	expensereport	Transaction	Full	X	X
Fair Value Price	fairvaluepricelist	List	Create, delete, and search are allowed. Copy is not allowed.		X
Folder	folder	File Cabinet	Full		X
Generic Resource	genericresource	Entity	Copy Not Allowed	X	X
Gift Certificate	giftcertificate	List	Full	X	X
Gift Certificate Item	giftcertificateitem	Item	Full	X	X
Global Account Mapping	globalaccountmapping	List	Full		X
Group	entitygroup	List	Full		X
Intercompany Allocation Schedule	intercompallocationschedule	List	Create, Read, Update, and Delete	X	X
Intercompany Journal Entry	intercompanyjournalentry	Transaction	Full	X	X

Record Name	Record ID	Record Category	Scripting Level	Scriptable in Client SuiteScript	Scriptable in Server SuiteScript
Intercompany Transfer Order	intercompanytransferorder	Transaction	Full	X	X
Inventory Adjustment	inventoryadjustment	Transaction	Full	X	X
Inventory Cost Revaluation	inventorycostrevaluation	Transaction	Full	X	X
Inventory Count	inventorycount	Transaction	Full	X	X
Inventory Detail	inventorydetail	Subrecord	See Scripting the Inventory Detail Subrecord.	X	Server-side scripts must access through the parent record
Inventory Item (Also referred to in the UI as Inventory Part)	inventoryitem	Item	Full	X	X
Inventory Number	inventorynumber	List	Copy, Create, and Delete Not Allowed		X
Inventory Transfer	inventorytransfer	Transaction	Full		X
Invoice	invoice	Transaction	Full	X	X
Issue	issue	Support	Full	X	X
Item Account Mapping	itemaccountmapping	List	Full		X
Item Search	item	Item	Search Only		
Item Demand Plan	itemdemandplan	Transaction	Copy Not Allowed, No Available Transforms	X	X
Item Fulfillment	itemfulfillment	Transaction	Copy and Create Not Allowed	X	X
Item Group	itemgroup	Item	Full	X	X
Item Receipt	itemreceipt	Transaction	Copy and Create Not Allowed	X	X
Item Revision	itemrevision	List	Full		X
Item Supply Plan	itemsupplyplan	Transaction	Full	X	X
Journal Entry	journalentry	Transaction	Full	X	X
Kit	kititem	Item	Full	X	X
Landed Cost	landedcost	Subrecord	Full		X
Lead	lead	Entity	Full	X	X
Location	location	List	Full		X
Lot Numbered Assembly Item	lotnumberedassemblyitem	Item	Full	X	X
Lot Numbered Inventory Item	lotnumberedinventoryitem	Item	Full	X	X
Manufacturing Cost Template	manufacturingcosttemplate	List	Full		X
Manufacturing Planned Time	mfgplannedtime	Transaction	Search Only	X	X
Manufacturing Operation Task	manufacturingoperatortask	Transaction	Copy Not Allowed	X	X
Manufacturing Routing	manufacturingrouting	List	Full		X
Markup	markupitem	Item	Full	X	X
Message	message	Communication	For details, see <a href="#">Message</a> .		X
Multi-Book Accounting Transaction	accountingtransaction	Transaction	Search Only	X	X
Nexus	nexus	List	Copy Not Allowed		X
Non-Inventory Part	noninventoryitem	Item	Full	X	X
Note	note	Communication	Full		X
Opportunity	opportunity	Transaction	Full	X	X
Order Schedule	orderschedule	Subrecord	See <a href="#">Working with Subrecords in SuiteScript</a>	X	Server-side scripts must access

Record Name	Record ID	Record Category	Scripting Level	Scriptable in Client SuiteScript	Scriptable in Server SuiteScript
					through the parent record
Other Charge Item	otherchargeitem	Item	Full		
Other Name	othername	Entity	Full		X
Partner	partner	Entity	Full	X	X
Paycheck	paycheck	Transactions	Read and edit only.	X	X
Paycheck Journal	paycheckjournal	Transaction	Full	X	X
Payment	paymentitem	Item	Full	X	X
Payroll Batch	payroll batch	Transactions	Full	X	X
Payroll Batch Employee	payrollbatchaddemployees	Transactions	Read and edit only.	X	X
Payroll Item	payrollitem	List	Full	X	X
Phone Call	phonecall	Activity	Full	X	X
Price Level	pricellevel	Lists	Search Not Available	X	X
Project (Job)	job	Entity	Full	X	X
Project Expense Type	projectexpensetype	List	Full		X
Project Task	projecttask	Event	Full	X	X
Project Template	projecttemplate	Entity	Copy Not Allowed	X	X
Promotion	promotioncode	Marketing	Full	X	X
Prospect	prospect	Entity	Full	X	X
Purchase Contract	purchasecontract	Transaction	Full	X	X
Purchase Order	purchaseorder	Transaction	Full	X	X
Reallocate Items	reallocatetitem	Item	Exposed for user event scripts only. This record can not be created, loaded, or deleted using scripts.		
Requisition	purchaserequisition	Transaction	Full	X	X
Resource Allocation	resourceallocation	Activities	Full	X	X
Return Authorization	returnauthorization	Transaction	Full	X	X
Revenue Arrangement	revenuearrangement	Transaction	Full		X
Revenue Commitment	revenuecommitment	Transaction	Create Not Allowed. See Usage Notes.	X	X
Revenue Commitment Reversal	revenuecommitmentreversal	Transaction	Create Not Allowed. See Usage Notes.	X	X
Revenue Recognition Plan	revrecplan	List	Copy, create, and delete are not allowed.		X (User Event Scripts Not Supported)
Revenue Recognition Schedule	revrecschedule	List	Copy, Create, and Delete Not Allowed		X (User Event Scripts Not Supported)
Revenue Recognition Template	revrectemplate	List	Full		X (User Event Scripts Not Supported)
Role	role	List	Search Only		X
Sales Order	salesorder	Transaction	Full	X	X
Sales Tax Item	salestaxitem	List	Full		X
Scheduled Script Instance		Customization	Search Only		X
Script	Need Internal ID. See Script description	Customization	Copy, Create and Delete Not Allowed. See Usage Notes		X
Script Deployment	Need Internal ID. See Script Deployment description	Customization	See Usage Notes		X

Record Name	Record ID	Record Category	Scripting Level	Scriptable in Client SuiteScript	Scriptable in Server SuiteScript
Serialized Assembly Item	serializedassemblyitem	Item	Full	X	X
Serialized Inventory Item	serializedinventoryitem	Item	Full	X	X
Service	serviceitem	Item	Full	X	X
Shipping Item	shipitem	Item	Create and Copy Not Allowed		X
Solution	solution	Support	Full	X	X
Statistical Journal Entry	statisticaljournalentry	Transaction	Full	X	X
Subsidiary	subsidiary	List	Full	X	X
Subtotal	subtotalitem	Item	Full	X	X
Task	task	Activity	Full	X	X
Tax Control Account	taxacct	List	Delete and Search Not Allowed	X	X
Tax Group	taxgroup	List	Full		X
Tax Period	taxperiod	List	Full		X
Tax Type	taxtype	List	Full		X
Term	term	List	Search Not Available	X	X
Time	timebill	Transaction	Full	X	X
Topic	topic	Support	Full		X
Transaction Search	transaction	Transaction	Search Only	X	X
Transfer Order	transferorder	Transaction			
Unit of Measure	unitstype	List	Full		X
Vendor	vendor	Entity	Full	X	X
Vendor Bill	vendorbill	Transaction	Full	X	X
Vendor Category	vendorcategory	List	Search Not Available	X	X
Vendor Credit	vendorcredit	Transaction	Full	X	X
Vendor Payment	vendorpayment	Transaction	Full	X	X
Vendor Return Authorization	vendorreturnauthorization	Transaction	Full	X	X
Web Site Setup	website	Website	Not supported in beforeLoad user event scripts.		X
Work Order	workorder	Transaction	Full	X	X
Work Order Close	workorderclose	Transaction	Copy and Create Not Allowed. You must use nlapiTransformRecord to "create" a new instance of this record.		X
Work Order Completion	workordercompletion	Transaction	Copy and Create Not Allowed. You must use nlapiTransformRecord to "create" a new instance of this record.		X
Work Order Issue	workorderissue	Transaction	Copy and Create Not Allowed. You must use nlapiTransformRecord to "create" a new instance of this record.		X
Workplace	workplace	List	Full	X	X



**Important:** Each custom list and custom record will have a unique internal ID. For example, a custom record's internal ID might be `customrecord22` or `customrecord5`. A custom list's ID might be `customlist7` or `customlist_shirtcolors`, depending on whether you have accepted the default ID assigned to the custom list (for example, `customlist7`) or you have created your own customized script ID (for example, `customlist_shirtcolors`).

To see a list of IDs for all your custom records, in the UI go to Customization > Lists, Records, & Fields > Record Types. For custom list IDs, go to Customization > Lists, Records, & Fields > Lists. If you have the Show Internal IDs preference enabled, all internal IDs will appear in the ID column. To enable the Show Internal IDs preference, go to Home > Set Preferences > click the Show Internal IDs check box > click Save.

# Activities

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following activity records are scriptable in SuiteScript:

- Activity
- Event
- Phone Call
- Project Task
- Resource Allocation
- Task
- Work Calendar

## Activity

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `activity`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Event

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `calendarevent`. Note that setting recurring events in SuiteScript is not currently supported.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Phone Call

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `phonecall`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Project Task

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The project task record can be used to keep track of specific activities and milestones associated with a project.

The internal ID for this record is `projecttask`.

The project task record is available when the Project Management feature is enabled at Setup > Company > Enable Features, on the Company subtab. When the feature is enabled, you can access the project task record in the UI by navigating to an existing project and clicking the New Project Task or New Milestone button. For details on working with this record in the UI, see the help topic [Project Tasks](#).

Project task records cannot be created as standalone records. Rather, you create a project task for a specific project record, and the task remains attached to that record. For information on working with the project record in SuiteScript, see [Project \(Job\)](#).

## Project Tasks Versus Milestone Tasks

Every project task record is designated as either a project task or a milestone task. While a project task is used to represent an activity, a milestone task is used to represent a checkpoint in the overall progress of the project.

Although they are the same type of record, a milestone task cannot have values in the `estimatedwork` body or sublist fields. Therefore, if you create a project task record and do not include any estimated work, the record is automatically saved as a milestone task. If you do include estimated work, the record is saved as a project task.

These same rules apply to the updating of records as well. In other words:

- To convert a project task into a milestone task, clear the `estimatedwork` body field and all values from the `Assignees` sublist.
- To convert a milestone task into a project task, add a value to the `estimatedwork` body field or add at least one record to the `Assignees` sublist, with a positive value of estimated work.

Note that the `estimatedwork` body field is populated by the sum of estimated work listed for `Assignees`. If you explicitly set a value for the `estimatedwork` body field, and you also include `Assignees`, the value you specify for the body field is overwritten based on the sublist's `estimatedwork` values. If you do not include `Assignees`, you can explicitly assign a value to the `estimatedwork` body field.

## Supported Script Types

The project record is scriptable in server SuiteScript only.



All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Field Definitions

When creating new project tasks you must set the Project (*company*) field to the project/job ID. Project tasks are not standalone records, and therefore must be associated with a specific project.

For other details on body fields and sublist fields, See the [SuiteScript Records Browser](#), which lists all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Sample

The following example shows how you might create both a project task and a milestone task.

```
// create a project
var project = nlapiCreateRecord('job');
project.setFieldValue('companynamne', 'Reconstruction');
var projectId = nlapiSubmitRecord(project);

// create a project task
var task = nlapiCreateRecord('projecttask');
task.setFieldValue('estimatedwork', 2);
task.setFieldValue('title', 'Remove old furniture');
task.setFieldValue('company', projectId);
var task1Id = nlapiSubmitRecord(task);

// create another task depending on the first one with Finish-To-Start dependency
task = nlapiCreateRecord('projecttask');
task.setFieldValue('estimatedwork', 5);
task.setFieldValue('title', 'Paint walls');
task.setFieldValue('company', projectId);
task.selectNewLineItem('predecessor');
task.setCurrentLineItemValue('predecessor', 'task', task1Id);
task.setCurrentLineItemValue('predecessor', 'type', 'FS');
```

```

task.commitLineItem('predecessor');
var task2Id = nlapiSubmitRecord(task);

// create a milestone
task = nlapiCreateRecord('projecttask');
task.setFieldValue('estimatedwork', 0);
task.setFieldValue('title', 'Verify painting after the walls dry out');
task.setFieldValue('company', projectId);
task.selectNewLineItem('predecessor');
task.setCurrentLineItemValue('predecessor', 'task', task2Id);
task.setCurrentLineItemValue('predecessor', 'type', 'FS');
task.commitLineItem('predecessor');
var milestoneId = nlapiSubmitRecord(task);

```

## Resource Allocation

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

**Important:** For information on the availability of the Resource Allocations feature, please contact your account representative.

The resource allocation record supports reserving an employee's time for a particular project.

The internal ID for this record is `resourceallocation`.

In the UI, you access this record by going to Activities > Scheduling > Resource Allocations. Alternatively, you can view the resource allocations for a specific project through the project record's Resources subtab (Lists > Relationships > Projects, or Lists > Relationships > Jobs).

This record is available only if the Resource Allocations feature has been enabled at Setup > Company > Enable Features, on the Company tab. Note that this option will not be visible unless your account has been provisioned for this feature. For more details, contact your account representative.

## Supported Script Types

This record is scriptable in both client SuiteScript and server SuiteScript

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
<code>nlapiCopyRecord</code>	yes
<code>nlapiCreateRecord</code>	yes

Function	Supported?
nlapiDeleteRecord	yes
nlapiLoadRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Field Definitions

To create a new resource allocation record, you must reference two existing NetSuite records, as follows:

- project — A reference to a project record defined at Lists > Relationships > Projects. Note that in some NetSuite accounts, projects are referred to as jobs.
- allocationresource — A reference to an employee record, defined at Lists > Employees > Employees, for which the Project Record option has been checked. The Project Resource box is located on the Human Resources tab of the employee record.

You are also required to provide values for several other fields. Refer to the [SuiteScript Records Browser](#) for the internal IDs of all fields associated with this record.

Note also that numberhours and percentoffime are read-only fields. They are returned when you load the record, but they cannot be modified.

For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).

## Code Samples

The following example shows how to create a resource allocation record:

```

var AMOUNT = "10.0";
var PROJECT1 = "43";
var RESOURCE1 = "45";
var ALLOCATIONTYPE1 = "1"; // Hard
var ALLOCATIONTYPE2 = "2"; // Soft
var ALLOCATIONUNIT1 = "H"; // Hours
var ALLOCATIONUNIT2 = "P"; // Percent of Time
var NOTES1 = "My notes 1";
var STARTDATE1 = "4/20/2013";
var ENDDATE1 = "4/28/2013";

var record = nlapiCreateRecord('resourceallocation');
record.setFieldValue('allocationamount', AMOUNT);
record.setFieldValue('allocationresource', RESOURCE1);
record.setFieldValue('allocationtype',ALLOCATIONTYPE1 );
record.setFieldValue('allocationunit', ALLOCATIONUNIT1);
record.setFieldValue('startdate', STARTDATE1);
record.setFieldValue('notes', NOTES1);

```

```
record.setFieldValue('project', PROJECT1);
record.setFieldValue('enddate', ENDDATE1);
var recId = nlapiSubmitRecord(record);
```

## Task



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is task.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Work Calendar



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is workcalendar.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

# Entities



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following entity records are scriptable in SuiteScript:

- Competitor
- Contact
- Customer
- Employee
- Entity
- Generic Resource
- Lead
- Other Name
- Partner
- Project (Job)
- Project Template
- Prospect
- Vendor

## Competitor



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is competitor.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Contact



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is contact.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Customer



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is **customer**.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

### Notes on Scripting Customer Fields

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
datecreated	Date Created	<p>This is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field.</p> <p>Tip: If you need to capture "date created" information that is not related to the date the record was created in NetSuite, create a custom field and set it to auto-default to today's date.</p>
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
<b>Search Filters and Search Columns</b>		
ccnumber	Credit Card Number	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Notes on Scripting Customer Sublists

You can update the contactaccessroles sublist to provide Customer Center access to contacts. You can provide access to contacts that already exist in NetSuite and that have already been attached to a customer that already exists in NetSuite. The workflow is as follows: 1) Add customer. 2) Add contacts. 3) Attach contacts to customer. 4) Update customer with contact access information.

The fields in this sublist map to the fields on the System Information, Access subtab in the UI. These fields include: a Boolean field that indicates whether a contact has access to NetSuite, contact name key field, email address used to log in to NetSuite, password used to log in to NetSuite, NetSuite role (Customer Center), and a Boolean field that indicates whether the contact should receive a notification email when access changes are made. If this Notify field is set to true, an email is sent.

## Transform Types

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

Target Record Type	Target Record Internal ID	Field Defaults
Cash Sale	cashsale	billdate
Customer Payment	customerpayment	
Quote	estimate	
Invoice	invoice	billdate
Opportunity	opportunity	
Sales Order	salesorder	

## Employee

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is employee.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in

Field Internal ID	Field UI Label	Note
		beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Transform Types

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

Target Record Type	Target Record Internal ID	Field Defaults
Expense Report	expensereport	
Time	timebill	

## Entity

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `entity`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).

## Generic Resource

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `genericresource`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Use a generic resource as a placeholder for resource allocation and project task assignment. This record is primarily used when a specific resource is not available. See the help topic [Generic Resources](#) for additional information.

## Supported Script Types

The generic resource record is supported in all client and server-side scripts.

# Supported Functions

The following SuiteScript functionality is supported:

- Read
- Create
- Edit
- Delete
- Search

 **Note:** Copy is not supported.

## Lead

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is lead.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
datecreated	Date Created	<p>This is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field.</p> <p>Tip: If you need to capture “date created” information that is not related to the date the record was created in NetSuite, create a custom field and set it to auto-default to today’s date.</p>
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

Field Internal ID	Field UI Label	Note
<b>Search Filters and Search Columns</b>		
ccnumber	Credit Card Number	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Transform Types

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

Target Record Name	Target Record Internal ID	Field Defaults
Opportunity	opportunity	

## Other Name

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is othername.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Partner

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is partner.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

### Notes on Scripting Partner Fields

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Notes on Scripting Partner Sublists

You can update the contactaccessroles sublist to provide Partner Center access to contacts. You can provide access to contacts that already exist in NetSuite and that have already been attached to a partner that already exists in NetSuite. The workflow is as follows: 1) Add partner. 2) Add contacts. 3) Attach contacts to partner. 4) Update partner with contact access information.

The fields in this sublist map to the fields on the System Information, Access subtab in the UI. These fields include: a Boolean field that indicates whether a contact has access to NetSuite, contact name key field, email address used to log in to NetSuite, password used to log in to NetSuite, NetSuite role (Partner Center), and a Boolean field that indicates whether the contact should receive a notification email when access changes are made. If this Notify field is set to true, an email is sent.

## Project (Job)

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the project record to manage company initiatives.

The internal ID for this record is job.

To use the project record, you must have the Projects feature enabled at Setup > Company > Enable Features, on the Company tab. If you plan to do advanced project tracking, you must also enable Project Management. If you do not see the Project Management check box, your company must first purchase the Project Management add-on from NetSuite.

To access the project record in the UI, choose Lists > Relationships > Projects (or Jobs). For help working with projects manually, see the help topic [Projects](#).

## Supported Script Types

The project record is scriptable in both client and server SuiteScript.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.



Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes

## Field Definitions

Note that the datecreated body field is a system-generated field that marks the date the record was created in NetSuite. You cannot change or override this field. If you need to capture “date created” information that is not related to the date the record was created in NetSuite, create a custom field and set it to auto-default to today’s date.

For more details on available fields, see the [SuiteScript Records Browser](#), which lists all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Adding a Resource With Multiple Roles

When using SuiteScript, the behavior of the Resources sublist differs slightly from the behavior in the UI. Specifically, in the UI, each line must have a unique value in the Name field. Further, in the UI, you can specify more than one role for each resource.

NAME *	EMAIL	ROLE *
Henson, Max		Assistant Manager Project Manager Designer

With SuiteScript, if you want to add a resource that has two different roles, you set up your code as if you are adding two sublist records for that resource -- one for each role. The jobresource values are not required to be unique. The following example illustrates this technique. In this example, only two unique resources are being added, but because one resource has two roles, that resource is represented twice.

```

proj.setFieldValue('companynamename', 'Launch');
proj.setFieldValue('subsidiary', '1');

proj.selectNewLineItem('jobresources');
proj.setCurrentLineItemText('jobresources', 'jobresource', 'Employee Resource 1');
proj.setCurrentLineItemText('jobresources', 'role', 'Staff');
proj.commitLineItem('jobresources');

proj.selectNewLineItem('jobresources');
proj.setCurrentLineItemText('jobresources', 'jobresource', 'Employee Resource 1');
proj.setCurrentLineItemText('jobresources', 'role', 'Project Manager');
proj.commitLineItem('jobresources');

```

```
var id = nlapiSubmitRecord(proj);
var savedProj = nlapiLoadRecord('job', id);
savedProj.selectLineItem('jobresources', '1');
savedProj.setCurrentLineItemText('jobresources', 'jobresource', 'Employee Resource 2');
savedProj.commitLineItem('jobresources');
nlapiSubmitRecord(savedProj);
```

Note that after you complete the add operation, the Resources sublist in the UI looks the same as it would if you had manually added one line for the resource, with multiple roles specified on that line, as shown in the illustration above.

## Scripting Projects with Advanced Revenue Management

The following table lists the scriptable fields associated with the Project record and advanced revenue management.

Field	Type	Internal ID
Accounting Period	List/Record	accountingperiod
Comments	Text Area	comments
Percentage Complete	Percent	percent

Before you begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

For help working with this record in the user interface, see the help topic [Using Advanced Revenue Management for Projects](#).

## Project Template



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is projecttemplate.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Use a project template as a standard starting point for projects and project items. Each record instance is reusable. See the help topic [Project Templates](#) for information on using project templates in the UI.

## Supported Script Types

The project template record is supported in all client and server-side scripts.

## Supported Functions

The following SuiteScript functionality is supported:

- Read
- Create
- Edit
- Delete
- Search



**Note:** Copy is not supported.

## Prospect



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is prospect.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
<b>Search Filters and Search Columns</b>		
ccnumber	Credit Card Number	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Transform Types

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

Target Record Name	Target Record Internal ID	Field Defaults
Estimate/Quote	estimate	
Opportunity	opportunity	
Sales Order	salesorder	

## Vendor

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is vendor.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
password	Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
password2	Confirm Password	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

## Transform Types

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

Target Record Name	Target Record Internal ID	Field Defaults
Purchase Order	purchaseorder	
Vendor Bill	vendorbill	
Vendor Payment	vendorpayment	

# Items



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following item records are scriptable in SuiteScript:

- Assembly Item
- Description
- Discount
- Download Item
- Gift Certificate Item
- Inventory Item
- Item Search
- Item Group
- Kit
- Lot Numbered Assembly Item
- Lot Numbered Inventory Item
- Markup
- Non-Inventory Part
- Other Charge Item
- Payment
- Reallocate Items
- Serialized Assembly Item
- Serialized Inventory Item
- Service
- Shipping Item
- Subtotal

## Using Item Records in SuiteScript



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section includes the following topics:

- Loading Item Types
- Filtering Items by Type
- Advanced Revenue Management Scripting with Items

For information about working with items in the UI, see the help topics [Using Item Records](#) and [Item Types](#).

## Loading Item Types

When using `nlapiLoadRecord(type, id, initializeValues)`, 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, lotnumberedassemblyitem, serializedassemblyitem

## Filtering Items by Type

The following are valid search filter item type IDs. Note that the item filter IDs are case-sensitive.

### Item Type IDs

Assembly	Markup
Description	NonInvtPart
Discount	OthCharge
DwnLdItem	Payment
EndGroup	Service
GiftCert	ShiptItem
Group	Subtotal
InvtPart	TaxGroup
Kit	TaxItem

To use these IDs:

1. Create a script that will search for items of a specific type or types (for example, search for all non-inventory items).
2. Next, see any of the valid SuiteScript item type IDs.

### Sample Code

```
//Create a script that will search for all non-inventory part items
function searchnoninventpart()
{
var filters = new Array();
filters[0] = new nlobjSearchFilter('type', null, 'anyof', 'NonInvtPart');
var columns = new Array();
columns[0] = new nlobjSearchColumn('internalId');
var items = nlapiSearchRecord('item', null, filters, columns);
}
```

## Advanced Revenue Management Scripting with Items

The item record (internal ID item) contains several additional accounting fields associated with the Advanced Revenue Management feature. The following table lists these scriptable fields.

Field	Type	Internal ID



Create Revenue Plans On	List/Record	createrevenueplanson
Item Revenue Category	List/Record	itemrevenuecategory
Revenue Allocation Group	List/Record	revenueallocationgroup
Revenue Recognition Rule	List/Record	revenuerecognitionrule

Before you begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

For help working with this record in the user interface, see the help topic [Configuring Items for Advanced Revenue Management](#).

## Assembly Item

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `assemblyitem`. This record is also sometimes referred to as Build/Assembly.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

For information about using build/assembly items in the UI, see the help topic [Assembly Items](#).

The following sample changes the item name of an assembly.

```
var assembly = nlapiLoadRecord('assemblyitem', 123);
assembly.setFieldValue('itemid', 'new name');
nlapiSubmitRecord(assembly);
```

## Description

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `descriptionitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using description items in the UI, see the help topic [Description Items](#).



## Discount

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is discountitem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using discount items in the UI, see the help topic [Discount Items](#).

## Download Item

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is downloaditem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using download items in the UI, see the help topic [Download Items](#).

## Gift Certificate Item

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is giftcertificateitem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using gift certificate items in the UI, see the help topic [Gift Certificates](#).

## Inventory Item

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is inventoryitem.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using inventory items in the UI, see the help topic [Inventory Items](#).

## Item Group

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is itemgroup.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using item groups in the UI, see the help topic [Item Groups](#).

## Item Search

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is item. Note that the item search record is a **search record only**. You cannot create or copy this record.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Kit

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is kititem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using kit items in the UI, see the help topic [Kit/Package Items](#).



## Lot Numbered Assembly Item

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `lotnumberedassemblyitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using lot numbered items in the UI, see the help topic [Lot Numbered Items](#). For information about using assembly items in the UI, see the help topic [Assembly Items](#).

## Lot Numbered Inventory Item

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `lotnumberedinventoryitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using lot numbered items in the UI, see the help topic [Lot Numbered Items](#). For information about using inventory items in the UI, see the help topic [Inventory Items](#).

## Markup

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `markupitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using markup items in the UI, see the help topic [Markup Items](#).

## Non-Inventory Part

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `noninventoryitem`.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using non-inventory items in the UI, see the help topic [Non-InVENTORY Items](#).

## Other Charge Item

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is otherchargeitem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using other charge items in the UI, see the help topic [Other Charge Items](#).

## Payment

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is paymentitem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using payment items in the UI, see the help topic [Payment Items](#).

## Reallocate Items

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is reallocateitem.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

This record can not be created, loaded, or deleted using scripts.



This record is only exposed in user event scripts. You can execute beforeLoad, beforeSubmit, and afterSubmit user event scripts on this record.

For information about working with this type of item in the UI, see the help topic [Reallocating Items](#).

## Serialized Assembly Item

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `serializedassemblyitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using serialized items in the UI, see the help topic [Serial Numbered Items](#). For information about using assembly items in the UI, see the help topic [Assembly Items](#).

## Serialized Inventory Item

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `serializedinventoryitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using serialized items in the UI, see the help topic [Serial Numbered Items](#). For information about using inventory items in the UI, see the help topic [Inventory Items](#).

## Service

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `serviceitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using service items in the UI, see the help topic [Service Items](#).



# Shipping Item

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `shipitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

A shipping item is a delivery method for a shipping carrier. It describes how to ship an item and can include shipping rate information, handling rates, and rules for shipping and handling. It can also specify when shipping is free. See the help topic [Shipping Items](#) for additional information.

## Supported Script Types

The shipping item record is supported in all server-side scripts. Client-side scripting is not supported.

## Supported Functions

The following SuiteScript functionality is supported:

- Read
- Edit
- Delete
- Search

**Note:** Create and copy are not supported. You can create this record only in the UI.

## Usage Notes

The following shipping item record components are not scriptable:

- Handling Table
- Shipping Table
- Free Shipping Items Tab

## Subtotal

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `subtotalitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.





**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For information about using subtotal items in the UI, see the help topic [Subtotal Items](#).

# Communications

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following communication records are scriptable in SuiteScript:

- Message
- Note

## Message

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is **message**.

For new messages, full scripting is supported. This support allows you to do the equivalent of clicking the **Attach** button in the UI for email messages. Note that there are also other types of messages, but those are not supported. In the message that you are creating you can set all supported fields for email messages before saving. Once you save the new message, there are restrictions applied to it. You cannot edit the existing message. It is possible to delete an existing message, if the permissions for the user are set to allow this type of operation. However, this operation is not recommended because it can cause email communication threads to be displayed incorrectly.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## User Notes

Message records can only be edited during the create operation. After they are created and submitted, existing message records cannot be edited.

Existing message records can be copied and deleted.

Only beforeLoad and afterSubmit user event scripts will execute on the Message record type when a message is created by an inbound email case capture. Scripts set to execute on a beforeSubmit event will not execute.

For example, if you have a test script like the following deployed to the Message record type:

```
function beforeLoad(type, name)
{
    nlapiLogExecution('DEBUG', 'Before Load');
}
function beforeSubmit(type, name)
{
    nlapiLogExecution('DEBUG', 'Before Submit');
}
function afterSubmit(type, name)
```

```
{  
    nlapiLogExecution('DEBUG', 'After Submit');  
}
```

only the beforeLoad(...) and afterSubmit(...) functions will execute if the message was created to respond to an emailed case.

## Note



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is note.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

# Transactions



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following transaction records are scriptable in SuiteScript.

- Assembly Build
- Assembly Unbuild
- Bin Putaway Worksheet
- Bin Transfer
- Blanket Purchase Order
- Cash Refund
- Cash Sale
- Charge
- Check
- Credit Card Charge
- Credit Card Refund
- Credit Memo
- Custom Transaction
- Customer Deposit
- Customer Payment
- Customer Refund
- Deposit
- Deposit Application
- Estimate / Quote
- Expense Report
- Intercompany Journal Entry
- Intercompany Transfer Order
- Inventory Adjustment
- Inventory Cost Revaluation
- Inventory Count
- Inventory Detail
- Inventory Transfer
- Invoice
- Item Demand Plan
- Item Fulfillment
- Item Receipt
- Item Supply Plan
- Journal Entry

- Landed Cost
- Manufacturing Operation Task
- Manufacturing Planned Time
- Multi-Book Accounting Transaction
- Opportunity
- Order Schedule
- Paycheck
- Paycheck Journal
- Payroll Batch
- Payroll Batch Employee
- Purchase Contract
- Purchase Order
- Requisition
- Return Authorization
- Revenue Arrangement
- Revenue Commitment
- Revenue Commitment Reversal
- Sales Order
- Statistical Journal Entry
- Time
- Transaction Search
- Transfer Order
- Vendor Bill
- Vendor Credit
- Vendor Payment
- Vendor Return Authorization
- Work Order
- Work Order Close
- Work Order Completion
- Work Order Issue



**Warning:** Memorized transactions do not support SuiteScript.

## Assembly Build



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is assemblybuild.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Assembly Unbuild

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `assemblyunbuild`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Bin Putaway Worksheet

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `binworksheet`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Copy and Update are not allowed for this record.

This record can only be scripted in dynamic mode. For details about dynamic scripting, see the following help topics:

- [Working with Records in Dynamic Mode](#)
- [How do I enable dynamic mode?](#)
- [Is dynamic mode better than standard mode?](#)
- [Standard vs. Dynamic Mode Code Samples](#)

Note that client (remote object) scripting does not support dynamic scripting.



## Bin Transfer



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is bintransfer.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

## Blanket Purchase Order



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

With this record, you can take advantage of fixed pricing for a preset number of items that you will buy during a specific time period. This approach lets you avoid sporadic pricing negotiations with vendors. The internal ID of this record is blanketpurchaseorder.

This record is available only when the Blanket Purchase Order feature is enabled at Setup > Company > Enable Features, on the Transactions subtab.

In the user interface, you access this record at Transactions > Purchases > Enter Blanket Purchase Order. For help working with this record in the user interface, see the help topic [Creating a Blanket Purchase Order](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes



Function	Supported?
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Cash Refund

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `cashrefund`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
<code>ccnumber</code>	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
<code>estgrossprofit</code>	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
<code>estgrossprofitpercent</code>	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.



Field Internal ID	Field UI Label	Note
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

## Cash Sale



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `cashsale`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
ccnumber	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in <code>beforeSubmit</code> user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
estgrossprofit	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
estgrossprofitpercent	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
isrecurringpayment	Recurring Payment	A value for this field is stored only if the value for <code>paymentmethod</code> is a credit card.
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

# Charge



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The charge record is used to represent a single billable amount that a client must pay.

The internal ID for this record is charge.

The charge record is available only when the Charge-Based Billing feature is enabled at Setup > Company > Enable Features, on the Transactions subtab. When the feature is enabled, you can access the charge record in the UI by choosing Transactions > Customers > Create Charges > List.

## Supported Script Types

The charge record is scriptable in server SuiteScript only.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiSearchRecord	Yes
nlapiSubmitRecord	Yes
nlapiTransformRecord	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Sample

The following sample shows how to create a charge record.

```
var charge = nlapiCreateRecord('charge');
charge.setFieldText('stage', 'Ready for billing');
charge.setFieldValue('chargetype', 'Time');
charge.setFieldValue('billto', '43');
charge.setFieldValue('chargedate', '3/19/2013');
```

```

charge.setFieldValue('salesorder', '122');
charge.setFieldValue('salesorderline', '1');
charge.setFieldValue('currency', '1');
charge.setFieldValue('billingitem', '21');
charge.setFieldValue('timerecord', '2');
charge.setFieldValue('description', 'Charge description');
charge.setFieldValue('rate', '2');
charge.setFieldValue('quantity', '3');
charge.setFieldValue('amount', '5');
recId = nlapiSubmitRecord(charge);

```

## Check

**i Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is check.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
usertotal	Amount	This field is not available via search or lookup for any transactions.

## Using Landed Cost Fields

When you create a landed cost category, the associated field IDs for the first category are landedcostamount1 and landedcostsource1. If you create a second category, the IDs will be landedcostamount2 and landedcostsource2.

This pattern increments by one with each additional category. For example, the IDs for the next landed cost category will be landedcostamount3 and landedcostsource3, and so on.

## Credit Card Charge

**i Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is creditcardcharge.

The credit card charge record is available only when a credit card account exists in the system. For information about creating a credit card account, see the help topic [Creating Accounts](#).

The script user must have the Credit Card permission with Full permission level.



## Supported Script Types

Only server SuiteScript is supported. Scripting on the client is not supported for this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Supported Operations

This record is fully scriptable.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	yes

## Code Samples

The following code snippets show how to create a \$30 credit card charge.

```
var recordType = "creditcardcharge";

var record = nlapiCreateRecord(recordType, false)

record.setFieldValue("entity", "4");
record.setFieldValue("account", "128"); // Account must be created before (Type = Credit Card)
record.setFieldValue("exchangerate", "1.00");
record.setFieldValue("postingperiod", "45");
record.setFieldValue("memo", "MEMO");
record.setFieldValue("class", "1");
record.setFieldValue("department", "1");
record.setFieldValue("location", "1");

record.setLineItemValue("expense", "account", 1, "58");
record.setLineItemValue("expense", "amount", 1, "30.00");
record.setLineItemValue("expense", "memo", 1, "MEMO2");
record.setLineItemValue("expense", "class_", 1, "1");
record.setLineItemValue("expense", "department", 1, "1");
record.setLineItemValue("expense", "location", 1, "1");
record.setLineItemValue("expense", "customer", 1, "3");
record.setLineItemValue("expense", "isbillable", 1, "F");
```

```
nlapiSubmitRecord(record);
```

## Credit Card Refund

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is creditcardrefund.

The credit card refund record is available only when a credit card account exists in the system. For information about creating a credit card account, see the help topic [Creating Accounts](#).

The script user must have the Credit Card Refund permission with Full permission level.

## Supported Script Types

Only server SuiteScript is supported. Scripting on the client is not supported for this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Supported Operations

This record is fully scriptable.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	yes

## Code Samples

The following code snippets show how to create a \$30 credit card refund.

```
var recordType = "creditcardrefund";

var record = nlapiCreateRecord(recordType, false)

record.setFieldValue("entity", "4");
record.setFieldValue("account", "128"); // Account must be created before (Type = Credit Card)
record.setFieldValue("exchangerate", "1.00");
```



```

record.setFieldValue("postingperiod", "45");
record.setFieldValue("memo", "MEMO");
record.setFieldValue("class", "1");
record.setFieldValue("department", "1");
record.setFieldValue("location", "1");

record.setLineItemValue("expense", "account", 1, "58");
record.setLineItemValue("expense", "amount", 1, "30.00");
record.setLineItemValue("expense", "memo", 1, "MEMO2");
record.setLineItemValue("expense", "class_", 1, "1");
record.setLineItemValue("expense", "department", 1, "1");
record.setLineItemValue("expense", "location", 1, "1");
record.setLineItemValue("expense", "customer", 1, "3");
record.setLineItemValue("expense", "isbillable", 1, "F");

nlapiSubmitRecord(record);

```

## Credit Memo

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is creditmemo.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Customer Deposit

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is customerdeposit.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
ccnumber	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for

Field Internal ID	Field UI Label	Note
		external role users (for example, shoppers, online form users (anonymous users), customer center).
isrecurringpayment	Recurring Payment	A value for this field is stored only if the value for paymentmethod is a credit card.

## Customer Payment



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `customerpayment`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
ccnumber	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
isrecurringpayment	Recurring Payment	A value for this field is stored only if the value for paymentmethod is a credit card.

## Customer Refund



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `customerrefund`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
ccnumber	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
estgrossprofit	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
estgrossprofitpercent	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

## Deposit

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the deposit record to adjust the balance of an account.

The internal ID of this record is deposit.

In the UI, you access this record at Transactions > Bank > Make Deposits.

## Supported Script Types

The deposit record is scriptable in both client and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

The deposit record cannot be copied, but otherwise it is fully scriptable. For more details, see the following table.

Function	Supported?
nlapiCopyRecord	No
nlapiCreateRecord	Yes



Function	Supported?
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiSubmitRecord	Yes
nlapiTransformRecord	No

## Usage Notes

Be aware of the following:

- To successfully add a new deposit record, you must include at least one line in one of the following three sublists: Payments, Other Deposits, or Cash Back.
- If you use the Multi-Book Accounting feature, be aware that the Accounting Book Detail sublist is read-only and not scriptable.
- The deposit record does not support dynamic mode.

## Code Samples

The following samples show how to create, load, and delete deposit records.

```
function createBankDeposit(cashSaleId){
    var rec = nlapiCreateRecord("deposit");
    rec.setFieldValue("account","1");
    rec.setFieldValue("subsidiary","1");
    rec.setLineItemValue("payment","id",1,cashSaleId);
    rec.setLineItemValue("payment","deposit",1,"T");
    var recId = nlapiSubmitRecord(rec);
    return recId;
}

function testDeposit()
{
    var cashSaleId = null;
    var depositId = null;
    try{
        cashSaleId = createCashSale(); // or just type internalId of valid Cash Sale
        depositId = createBankDeposit(cashSaleId);
        var deposit = nlapiLoadRecord("deposit",depositId);
    }
    finally{
        if(depositId!=null)
            nlapiDeleteRecord("deposit",depositId);
        if(cashSaleId!= null)
            nlapiDeleteRecord("cashSale",cashSaleId);
    }
}
```



# Deposit Application

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `depositapplication`.

In SuiteScript, you do not use the `napiCreateRecord(...)` to create a Deposit Application record. Deposit applications are always created when a Customer Deposit is applied to an invoice. The application can only be created by applying an open Customer Deposit from the Deposit sublist of the Customer Payment. On submit, the backend creates a deposit application in the amount applied.

You can use the `doc` field on the Apply sublist of the Customer Payment to get the internal ID of the deposit or invoice.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Estimate / Quote

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `estimate`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
<code>estgrossprofit</code>	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
<code>estgrossprofitpercent</code>	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
<code>totalcostestimate</code>	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.



Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Expense Report

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `expensereport`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Intercompany Journal Entry

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Intercompany journal entries are a specialized type of journal available specifically for OneWorld. An intercompany journal entry records debits and credits to be posted to ledger accounts for transactions between two subsidiaries. These records adjust the value of any set of accounts without the need for transactions such as invoices and bills.

The internal ID for this record is `intercompanyjournalentry`.

In the user interface, you can access this record at [Transactions > Financial > Make Intercompany Journal Entries](#).

If your account has the Multi-Book Accounting feature enabled, you can also work with *book specific* intercompany journal entry records, which in the user interface are available at [Transactions > Financial > Make Book Specific Intercompany Journal Entries](#). Although they have different entry forms, both book specific and regular intercompany journal entries are the same record type. Within SuiteScript, they are differentiated by the `accountingbook` field. In other words, a record that has a value set for `accountingbook` is book specific. Otherwise, the record is a regular intercompany journal entry.

For help working with this record in the user interface, see the help topics [Making Intercompany Journal Entries](#) and [Book Specific Intercompany Journal Entries](#).

## Supported Script Types

The intercompany journal entry record is scriptable in both client and server SuiteScript.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.



Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Sample

The following example shows how to create a book specific intercompany journal entry. The record is book specific because a value has been set for the accountingbook field.

```
var rec = nlapiCreateRecord('intercompanyjournalentry');
rec.setFieldValue('accountingbook', 2); // Setting a value for this field makes the record book
// specific.
rec.setFieldValue('subsidiary', 1);
rec.setFieldValue('tosubsidiary', 3);

rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'linesubsidiary', 1);
rec.setCurrentLineItemValue('line', 'account', 1);
rec.setCurrentLineItemValue('line', 'credit', '2.00');
rec.commitLineItem('line');
rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'linesubsidiary', 1);
rec.setCurrentLineItemValue('line', 'account', 2);
rec.setCurrentLineItemValue('line', 'debit', '2.00');
rec.commitLineItem('line');

rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'linesubsidiary', 3);
rec.setCurrentLineItemValue('line', 'account', 6);
rec.setCurrentLineItemValue('line', 'credit', '2.00');
rec.commitLineItem('line');
rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'linesubsidiary', 3);
rec.setCurrentLineItemValue('line', 'account', 149);
rec.setCurrentLineItemValue('line', 'debit', '2.00');
rec.commitLineItem('line');
```

```
var id = nlapiSubmitRecord(rec);
;
```

## Intercompany Transfer Order

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

In accounts using NetSuite OneWorld and the Multi-Location Inventory(MLI) feature, you can use the Intercompany Transfer Order transaction to move inventory from a location for one subsidiary to a location for another subsidiary .

The internal ID for this record is intercompanytransferorder.

In the user interface, you can access this record at Transactions > Inventory > Enter Intercompany Transfer Orders..

For help working with this record in the user interface, see the help topic [Intercompany Inventory Transfers - Non-Arm's Length](#).

## Supported Script Types

The intercompany transfer order record is scriptable in both client and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes, when the transaction type is "Transfer Order" and Intercompany = "Yes".
nlapiTransformRecord	Yes, can be transformed into Item Fulfillment or Item Receipt records.

## Usage Notes

Intercompany transfer orders are only available when transfer pricing is used, meaning the Use Item Cost as Transfer Cost, at Setup > Accounting > Preferences > Accounting Preferences (Administrator), on the Order Management subtab, must be disabled.



This record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Code Sample

The following example shows how to create an intercompany transfer order.

```
...
NLRecordObject incoTransfer = nlapiCreateRecord('intercompanytransferorder', true);
incoTransfer.setFieldValue('orderstatus', expectedValues.get(orderstatus));
incoTransfer.setFieldValue('subsidiary', expectedValues.get(subsidiary));
incoTransfer.setFieldValue('tosubsidiary', expectedValues.get(tosubsidiary));
incoTransfer.setFieldText('location', sourceLocationName1);
incoTransfer.setFieldText('transferlocation', destinationLocationName1);
incoTransfer.selectNewLineItem('item');
incoTransfer.setCurrentLineItemValue('item', 'Item.item', itemKey);
incoTransfer.setCurrentLineItemValue('item', 'Item.rate', rate);
incoTransfer.setCurrentLineItemValue('item','quantity', quantity);
incoTransfer.commitLineItem('item');
String key = nlapiSubmitRecord(incoTransfer);
...
```

## Inventory Adjustment

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `inventoryadjustment`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Inventory Cost Revaluation

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The inventory cost revaluation record is used to recalculate the value of items configured to use standard costing.

The internal ID for this record is `inventorycostrevaluation`.

This record is available when the Standard Costing feature is enabled at Setup > Company > Enable Features, on the Items & Inventory subtab. When the feature is enabled, you can access the inventory cost revaluation record in the UI by choosing Transactions > Inventory > Revalue Inventory Cost.

## Supported Script Types

The inventory cost revaluation record is scriptable in both client SuiteScript and Server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Sample

The following sample shows how to create and update inventory cost revaluation records.

```
function inventoryCostRevaluation(){
    var recId = null;
    var newId = null;
    var recType = 'INVENTORYCOSTREVALUATION';
    try{
        // Add
        var revaluationRec = nlapiCreateRecord(recType);
        revaluationRec.setFieldValue('subsidiary','1');
        revaluationRec.setFieldValue('item',itemId1); // Some assembly item
        revaluationRec.setFieldValue('account','1');
        revaluationRec.setFieldValue('location','1');
        revaluationRec.setLineItemValue('costcomponent', 'cost', 1, '2');
        revaluationRec.setLineItemValue('costcomponent', 'componentitem', 1, componentItemId1);
        // Some inv. item
        revaluationRec.setLineItemValue('costcomponent', 'quantity', 1, '3');

        recId = nlapiSubmitRecord(revaluationRec);
        var revaluationAddedRec = nlapiLoadRecord(recType, recId);
    }
}
```

```

// Update
// Note you cannot change subsidiary and item
revaluationAddedRec.setFieldValue('account','2');
revaluationAddedRec.setLineItemValue('costcomponent', 'quantity', 1, '5');
nlapiSubmitRecord(revaluationAddedRec);

var revaluationUpdatedRec = nlapiLoadRecord(recType, recId2);

// Copy
var copiedRecord = nlapiCopyRecord(recType, recId);
newId = nlapiSubmitRecord(copiedRecord);
copiedRecord = nlapiLoadRecord(recType,newId);

//Search columns
var arrSearchColumns = new Array();
arrSearchColumns[0] = new nlobjSearchColumn('memo');
arrSearchColumns[1] = new nlobjSearchColumn('subsidiary');

// Filters
var arrSearchFilters = new Array();
arrSearchFilters[0] = new nlobjSearchFilter('memo', null,'contains', 'em');

//Search
var arrSearchResults = nlapiSearchRecord(recType,null,arrSearchFilters,arrSearchColumns);

nlapiLogExecution('DEBUG', "+argumentscallee.name +' passed.");
}

catch (e) {
    nlapiLogExecution('ERROR', "+argumentscallee.name +' failed!',e);

}

finally{
    if(recId!=null){
        nlapiDeleteRecord(recType, recId);
    }
    if(newId!=null){
        nlapiDeleteRecord(recType, newId);
    }
}
}

```

## Inventory Count



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The inventory count record enables you to maintain better inventory accuracy and tighter control of assets.

The internal ID for this record is `inventorycount`.

The inventory count record is available only when the Inventory Count feature is enabled at Setup > Enable Features, on the Items & Inventory subtab. For an overview of the feature, see the help topic [Inventory Count](#).

In the user interface, you access the inventory count record at Transactions > Inventory > Enter Inventory Count.

## Supported Script Types

The inventory count record is scriptable in both client and server SuiteScript.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
<code>nlapiCopyRecord</code>	Yes
<code>nlapiCreateRecord</code>	Yes
<code>nlapiDeleteRecord</code>	Yes
<code>nlapiLoadRecord</code>	Yes
<code>nlapiSearchRecord</code>	Yes
<code>nlapiSubmitRecord</code>	Yes
<code>nlapiTransformRecord</code>	No

## Code Sample

The following example shows how to create an inventory count record.

```
var REC_TYPE = 'inventorycount';
var inventoryCount = nlapiCreateRecord(REC_TYPE);

inventoryCount.setFieldValue('subsidiary', 1);
inventoryCount.setFieldValue('location', 1);
inventoryCount.setFieldValue('tranid', '5');
inventoryCount.setFieldValue('trandate', '11/20/2013');
inventoryCount.setFieldText('account', 'Checking');
```

```

inventoryCount.setFieldText('department', 'Department US');
inventoryCount.setFieldText('class', 'Class US');
inventoryCount.setFieldValue('memo', 'SS memo');

inventoryCount.selectNewLineItem('item');
inventoryCount.setCurrentLineItemValue('item', 'item', 247);
inventoryCount.setCurrentLineItemValue('item', 'rate', 69.5);
inventoryCount.setCurrentLineItemValue('item', 'memo', 'Line memo');
inventoryCount.commitLineItem("item");

var inventoryCountId = nlapiSubmitRecord(inventoryCount);

```

## Inventory Detail

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this subrecord is `inventorydetail`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Inventory Detail is scriptable from both the body field and the line item.

Inventory Detail is considered a subrecord, represented by the `nlobjSubrecord` object in SuiteScript. For details on working with this subrecord type, see [Scripting the Inventory Detail Subrecord](#). For general details on working with subrecords, see [Working with Subrecords in SuiteScript](#).

## Inventory Transfer

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `inventorytransfer`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Invoice

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is invoice.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
estgrossprofit	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
estgrossprofitpercent	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Item Demand Plan

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is itemdemandplan.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

A demand plan records the expected future demand for an item based on previous or projected demand. When the Demand Planning feature is enabled, demand plans can be created for inventory items. When the Work Orders feature is also enabled, demand plans also can be created for assembly/BOM items. Demand plans can only be created for items that have a value of "Time Phased" for the **supplyreplenishmethod** field.

Each demand plan record includes:

- A set of body fields used to uniquely identify the demand plan, define the time period it covers, and indicate the time period it uses (monthly, weekly, or daily).
  - Body fields must be defined before matrix field values can be edited.
- A matrix of projected quantities per time period, similar to the matrix used for item pricing.
  - In a monthly demand plan, this matrix includes a row for each month in the time period, and one column with the projected quantity demand for each month.
  - In a weekly demand plan, this matrix includes a row for each week in the time period, and one column with the projected quantity demand for each week.
  - In a daily demand plan, this matrix includes a row for each week in the time period and seven columns with the projected quantity demand for each day of each week.
- Review the following table for details about Item Demand Plan body and matrix sublist fields. For more details and code samples, see [Demand Plan Detail Sublist](#).

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
subsidiary	Subsidiary	Required in OneWorld accounts.
location	Location	Required when the Multi-Location Inventory feature enabled.
item	Item	Required. Can only use items with supplyreplenishment method set to Time Phased.
units	Unit of Measure	Optional. Available when the Multiple Units of Measure feature is enabled.
memo	Memo	Optional.
startdate	Start Date	Optional. Defaults to the first day of the current year, for example for 2011, defaults to 1/1/2011.
enddate	End Date	Optional. Defaults to the last day of the current year, for example for 2011, defaults to 12/31/2011.

Field Internal ID	Field UI Label	Note
demandplancalendartype	View	Required. Valid values are MONTHLY, WEEKLY, or DAILY. (Must use all capital letters.)
<b>Matrix Fields</b>		
quantity	Quantity	<ul style="list-style-type: none"> <li>■ For monthly and weekly demand plans, each row has one quantity column.</li> <li>■ For daily demand plans, each row has seven quantity columns.</li> </ul>
startdate	Start Date	System-calculated, read-only values. <ul style="list-style-type: none"> <li>■ For monthly plans, the date of the first day of the month that the row represents.</li> <li>■ For weekly and daily plans, the date of the first day of the week that the row represents, based on the preference set for First Day of Week at Setup &gt; Company &gt; General Preferences.</li> </ul>
enddate	End Date	System-calculated, read-only values. <ul style="list-style-type: none"> <li>■ For monthly plans, the date of the last day of the month that the row represents.</li> <li>■ For weekly and daily plans, the date of the last(seventh) day of the week that the row represents, based on the preference set for First Day of Week at Setup &gt; Company &gt; General Preferences.</li> </ul>



**Note:** It is recommended that you work with the Item Demand Plan record in dynamic mode. See the help topic [Working with Records in Dynamic Mode](#).

## Item Fulfillment



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is itemfulfillment. Copy and create are not allowed for this record.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

When working with this record, you can set *pick*, *pack*, or *ship* as event trigger types that will execute your user event script. In the NetSuite Help Center, see [User Event Script Execution Types](#) for more information.

# Item Receipt



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

An item receipt transaction records the receipt of returned items from customers. This transaction updates the following information:

- Items on return authorizations are recorded as received.
- Inventory records are updated for the new stock levels.
- Inventory asset accounts are updated with the values of returned items.
- Status of the return is updated.

The item receipt transaction is available when the Advanced Receiving feature is enabled.

For more details about this type of transaction, see the help topics [Receiving a Customer Return](#) and [Handling Returned Items](#).

The internal ID of this record is `itemreceipt`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following usage notes are included for this record:

### Using Landed Cost Fields

When you create a landed cost category, the associated field IDs for the first category are `landedcostamount1` and `landedcostsource1`. If you create a second category, the IDs will be `landedcostamount2` and `landedcostsource2`.

This pattern increments by one with each additional category. For example, the IDs for the next landed cost category will be `landedcostamount3` and `landedcostsource3`, and so on.

### Creating Item Receipt Records

You cannot create standalone item receipts using SuiteScript. For example, the following will throw an error:

```
var ir = nlapiCreateRecord("itemreceipt");
```

To create an item receipt, you must use the `nlapiTransformRecord(...)` API, which transforms the data from one record type, **purchase order**, for example, into an item receipt. To create an item receipt, your code would be similar to the following:

```
function trans()
```

```
{
  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 idl = nlapiSubmitRecord(trecord, true);

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

## Item Supply Plan



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `itemsupplyplan`.

This record includes the [Orders Sublist](#).

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The item, location, and units body fields cannot be changed in update operations.

An item supply plan's receiptdate cannot be earlier than the orderdate.

The ordercreated field is read-only. It is set to True when an order is generated from an item supply plan.

## Code Samples

The following code creates an item supply plan.

```

function createItemSupplyPlanMinimal()
{
    var isp = nlapiCreateRecord('itemsupplyplan');
    isp.setFieldValue('subsidiary', 1);
    isp.setFieldValue('location', 1);
    isp.setFieldValue('item', 165);

    isp.setFieldValue('memo', 'memotest');
    isp.setFieldValue('unit', 3);

    isp.selectNewLineItem('order');
    isp.setCurrentLineItemValue('order','orderdate', '05/05/2012');
    isp.setCurrentLineItemValue('order', 'receiptdate', '5/8/2012');
    isp.setCurrentLineItemValue('order', 'quantity', 1);
    isp.setCurrentLineItemValue('order', 'ordertype', 'PurchOrd');

    isp.commitLineItem('order');

    var id = nlapiSubmitRecord(isp);
}

```

The following code updates an existing item supply plan.

```

function updateItemSupplyPlan()
{
    var isp = nlapiLoadRecord('itemsupplyplan', 3);
    isp.setFieldValue('memo','memotest2');

    isp.setLineItemValue('order', 'receiptdate', 4, '11/3/2012');
    var id = nlapiSubmitRecord(isp);
}

```

## Journal Entry



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the journal entry record to adjust balances in your ledger accounts without entering posting transactions.

The internal ID for this record is `journalentry`.

In the user interface, you access this record in the user interface at Transactions > Financial > Make Journal Entries.

If your account has the Multi-Book Accounting feature enabled, you can also work with book specific journal entry records, which are available in the user interface at Transactions > Financial > Make Book Specific Journal Entries. Although they have different entry forms, both book specific and regular intercompany journal entries are the same record type. Within SuiteScript, they are differentiated by

the accountingbook field. In other words, a record that has a value set for accountingbook is book specific. Otherwise, the record is a regular intercompany journal entry.

For help working with this record in the user interface, see the help topics [Making Intercompany Journal Entries](#) and [Book Specific Journal Entries](#).

## Supported Script Types

The journal entry record is scriptable in both client SuiteScript and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

With the Advanced Revenue Management feature, you can directly attach a revenue recognition plan to a book specific journal entry or an intercompany journal entry. Before you begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

The following table lists the scriptable field associated with this record and advanced revenue management.

Field	Type	Internal ID
End Date	Date	enddate
Revenue Recognition Rule	List/Record	revenuerecognitionrule
Start Date	Date	startdate

For help working with this record in the user interface, see the help topic [Creating Revenue Elements from Journal Entries](#)

## Code Sample

The following example shows how to create a book specific journal entry. The record is book specific because a value has been set for the accountingbook field.

```
var initvalues = new Array();
initvalues.bookje = 'T';
var rec = nlapiCreateRecord('journalentry', initvalues);
rec.setFieldValue('accountingbook', '2'); // Setting a value for this field makes the record book specific.
rec.setFieldValue('subsidiary', '4');
rec.setFieldValue('trandate', '5/16/2013');
rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'account', '6');
rec.setCurrentLineItemValue('line', 'credit', '2.00');
rec.commitLineItem('line');
rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line', 'account', '149');
rec.setCurrentLineItemValue('line', 'debit', '2.00');
rec.commitLineItem('line');
var id = nlapiSubmitRecord(rec);
var x = 0;
```

## Landed Cost



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The landed cost subrecord supports the Landed Cost Allocation per Line functionality, which is part of the Landed Cost feature. Landed costs typically include location-specific expenses such as customs duties and freight fees. The landed cost subrecord can be used in conjunction with several transactions: check, credit card charge, item receipt, and vendor bill. The purpose of the subrecord is to show the landing costs associated with a particular line in the parent transaction's Items sublist.

The internal ID for this subrecord is `landedcost`.

The subrecord is available only when the Landed Cost feature is enabled at Setup > Company > Setup Tasks > Enable Features, on the Items & Inventory subtab. For details on working with the subrecord in the user interface, see the help topic [Using Landed Cost Allocation Per Line on Transactions](#).



**Important:** Landed cost is considered a subrecord, not a record. Subrecords are represented by the `nlobjSubrecord` object in SuiteScript. For general details on working with subrecords, see [Working with Subrecords in SuiteScript](#).

## Supported Script Types

The landed cost subrecord is scriptable in server SuiteScript only. The user events are not supported.

## Supported Functions

The landed cost subrecord is fully scriptable, which means that it can be created, updated, copied, deleted, and searched using SuiteScript.

## Usage Notes

To script to the landed cost subrecord, both of the following must be true:

- The parent transaction's Landed Cost per Line option is checked.
- The sublist item being referenced has been configured to use the Track Landed Cost option. For details on configuring the item, see the help topic [Set Up Item Records for Landed Cost](#).

## Code Sample

The following snippets show how to create a landed cost subrecord and perform other basic tasks.

```
// Creating a landed cost subrecord
var purchaseOrder = nlapiCreateRecord('purchaseorder');
purchaseOrder.setFieldText('entity', 'Acme Medical Supply');
purchaseOrder.setLineItemValue('item', 'item', 1, inventoryItemId);

var purchaseOrderId = nlapiSubmitRecord(purchaseOrder);

var itemReceipt = nlapiTransformRecord('purchaseorder', purchaseOrderId, 'itemreceipt');
itemReceipt.selectLineItem('item', 1);
itemReceipt.setCurrentLineItemValue('item', 'location', 1);
itemReceipt.setFieldValue('landedcostperline', 'T');

var landedCost = itemReceipt.createCurrentLineItemSubrecord('item', 'landedcost');
landedCost.selectNewLineItem('landedcostdata');
landedCost.setCurrentLineItemValue('landedcostdata', 'costcategory', 1);
landedCost.setCurrentLineItemValue('landedcostdata', 'amount', 456);
landedCost.commitLineItem('landedcostdata');
landedCost.selectNewLineItem('landedcostdata');
landedCost.setCurrentLineItemValue('landedcostdata', 'costcategory', 3);
landedCost.setCurrentLineItemValue('landedcostdata', 'amount', 78.96);
landedCost.commitLineItem('landedcostdata');
landedCost.commit();

itemReceipt.commitLineItem('item');

var itemReceiptId = nlapiSubmitRecord(itemReceipt);

// Viewing the subrecord
itemReceipt = nlapiLoadRecord('itemreceipt', itemReceiptId);
itemReceipt.selectLineItem('item', 1);
landedCost = itemReceipt.viewLineItemSubrecord('item', 'landedcost', 1);
landedCost.getLineItemValue('landedcostdata', 'amount', i);
```

```

// Updating the subrecord
landedCost = itemReceipt.editCurrentLineItemSubrecord('item', 'landedcost');

landedCost.removeLineItem('landedcostdata', 2);

landedCost.setLineItemValue('landedcostdata', 'costcategory', 1, 2);
landedCost.setLineItemValue('landedcostdata', 'amount', 1, 3.98);
landedCost.selectNewLineItem('landedcostdata');
landedCost.setCurrentLineItemValue('landedcostdata', 'costcategory', 3);
landedCost.setCurrentLineItemValue('landedcostdata', 'amount', 103);
landedCost.commitLineItem('landedcostdata');
landedCost.commit();

itemReceipt.commitLineItem('item');

nlapiSubmitRecord(itemReceipt);

// Deleting the subrecord
itemReceipt.selectLineItem('item', 1);
itemReceipt.removeCurrentLineItemSubrecord('item', 'landedcost');
itemReceipt.commitLineItem('item');
nlapiSubmitRecord(itemReceipt);

```

## Manufacturing Operation Task



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is manufacturingoperationtask.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

## Usage Notes

To work with the manufacturing operation task record, the Manufacturing Routing and Work Center feature must be enabled at Setup > Company > Enable Features, on the Items & Inventory tab.

In the UI, the manufacturing operation task record is accessed by going to Transactions > Manufacturing > Manufacturing Operation Tasks. Alternatively, you can navigate to the Operations subtab of a WIP work order that uses the routing feature. The Operations subtab lists existing operation task records for that work order and allows you to create new operation task records.

Note these additional details:

- This record supports client and server-side scripting.
- All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.
- Refer to the table below for details on supported API functions.

Function	Supported?
nlapiCopyRecord	No
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

- To create a new manufacturing operation task record, you must reference a valid existing WIP work order, as shown in the [Code Sample](#) below.

## Code Sample

When creating a manufacturing operation task record, you must use initValues to reference a valid existing WIP work order. For example:

```
var initValues = new Array();
initValues.workorder = '65';
var task = nlapiCreateRecord('manufacturingoperationtask', initValues);
task.setFieldValue('title', 'Some title');
task.setFieldValue('operationsequence', 6);
task.setFieldValue('setuptime', 30);
task.setFieldValue('runrate', 20);
task.setFieldValue('manufacturingcosttemplate', '1');
task.setFieldValue('manufacturingworkcenter', '113');
var recId = nlapiSubmitRecord(task);
```

## Manufacturing Planned Time

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The Manufacturing Planned Time search type enables you to search for data displayed on the Planned Time subtab of work orders. This subtab is available only when the Show Planned Capacity on Work Orders preference is enabled.

The Planned Time subtab is used to display data about the amount of time being allocated to each work center per day for the work order. This data is derived from the associated operation task records.

You can set the Show Planned Capacity on Work Orders preference at Setup > Accounting > Accounting Preferences. The preference is available only when the Manufacturing Routing and Work Center feature is enabled.

The internal ID for this record is mfgplannedtime.

## Supported Script Types

This search is supported in both client and server SuiteScript.

## Usage Notes

Be aware of the following:

- This record is a search record only. You cannot create or copy this record.
- In the user interface, you can view this search by navigating to Reports > New Search and clicking Manufacturing Planned Time.

For help creating search scripts, see [Search Samples](#).

## Multi-Book Accounting Transaction



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

If your account has the Multi-Book Accounting feature enabled, you can use SuiteScript to search for transactions using accounting book as a search filter or a search column. To execute this type of search, you use the multi-book accounting transaction search record.

The internal ID for this record is `accountingtransaction`. Note that this record is a search record only. You cannot create or copy this record.

In the user interface, you can view the multi-book accounting transaction search by navigating to Reports > New Search and clicking Multi-Book Accounting Transaction.

## Supported Script Types

This record is supported in both client and server SuiteScript.

## Sample Code

The following example shows how to search for a specific transaction and include in the results details about the transaction's accounting book and foreign exchange rate, among other data.

```
var filters = new Array();
filters[0] = new nlobjSearchFilter( 'internalid', null, 'is', '18' );
// last parameter represents the actual internal ID of the transaction

var columns = new Array();
columns[0] = new nlobjSearchColumn( 'accountingbook' ).setSort(false);
columns[1] = new nlobjSearchColumn( 'line', 'transaction', null ).setSort(false);
columns[2] = new nlobjSearchColumn( 'account' );
columns[3] = new nlobjSearchColumn( 'amount' );
columns[4] = new nlobjSearchColumn( 'exchangerate' );

var searchresults = nlapiSearchRecord( 'accountingtransaction', null, filters, columns );

for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
{
```

```

var searchresult = searchresults[ i ];
var record = searchresult.getId( );
var rectype = searchresult.getRecordType( );
var book = searchresult.getValue( 'accountingbook' );
var line = searchresult.getValue( 'line', 'transaction' );
var account= searchresult.getValue( 'account' );
var amount= searchresult.getValue( 'amount' );
var fxrate= searchresult.getValue( 'exchangerate' );

alert(i + ": " + book + " | " + line + " | " + record + " | " + rectype + " | " + account + " |
" + amount + " | " + fxrate);
}

```

## Opportunity

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is opportunity.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
estgrossprofit	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
estgrossprofitpercent	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Order Schedule

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this subrecord is orderschedule.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Order Schedule is scriptable from both the body field and the line item.

Order Schedule is considered a subrecord, represented by the nlobjSubrecord object in SuiteScript. For details on working with subrecords, see [Working with Subrecords in SuiteScript](#).

## Paycheck

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is paycheck.

This record is available only when the Payroll, Payroll Service, and Enhanced Premier Payroll features are enabled at Setup > Company > Enable Features on the Employees subtab.

In the user interface, you can access this record at Transactions > Employees > Create Payroll. For help working with this record in the user interface, see the help topics [Calculating a Payroll Batch with Enhanced Premier Payroll](#) and [Editing an Individual Paycheck from a Payroll Batch](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

## Supported Functions

The paycheck record is read and edit only.

Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	no



Function	Supported?
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes

## Usage Notes

Be aware of the following:

- The paycheck record is created through the payroll batch record.

## Code Samples

The following code snippets show how to search for paycheck records.

```
var filters = new Array();
filters[0] = new nlobjSearchFilter('batchnumber', null, 'equalTo', 100);

// Define search columns
var columns = new Array();
columns[0] = new nlobjSearchColumn('batchnumber');
columns[1] = new nlobjSearchColumn('employee');

var searchResults = nlapiSearchRecord('paycheck', null, filters, columns);
```

## Paycheck Journal



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is paycheckjournal.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The Paycheck Journal feature must be enabled to work with the Paycheck Journal record.

The Paycheck Journal record is intended to enable global payroll solutions. You can use it along with the [Payroll Item](#) record to create custom payroll solutions and to support integrations with external payroll systems.

## Code Samples

The following sample creates a paycheck journal transaction.

```
//create Paycheck Journal with earnings and deduction sublist
// add 2 Earning and 1 deduction sublists

function createPaycheckJournal()
{
    var pj = nlapiCreateRecord('paycheckjournal');

    pj.setFieldValue('trandate', "6/10/2012");
    pj.setFieldValue('employee', 4 ); //internal Id of employee
    pj.setFieldValue('account', 28 ); //internal Id of account

    pj.selectNewLineItem('earning');
    pj.setCurrentLineItemValue('earning','payrollitem', '102');
    pj.setCurrentLineItemValue('earning', 'amount', 20.35);
    pj.commitLineItem('earning');

    pj.selectNewLineItem('earning');
    pj.setCurrentLineItemValue('earning','payrollitem', '102');
    pj.setCurrentLineItemValue('earning', 'amount', 33.35);
    pj.commitLineItem('earning');

    pj.selectNewLineItem('deduction');
    pj.setCurrentLineItemValue('deduction','payrollitem', '103');
    pj.setCurrentLineItemValue('deduction', 'amount', 444.44);
    pj.commitLineItem('deduction');

    nlapiSubmitRecord(pj);
}
```

The following sample updates a paycheck journal transaction.

```
//update Paycheck journal
//set new amount of line 2 Earning list
// and clear deduction list

function updatePaycheckJournal()
{
    var pj = nlapiLoadRecord('paycheckjournal', 305 ); // internalID of

    pj.setLineItemValue('earning', 'amount', 2, 444.44); // 2 is the line no we intend to updat
e

    for (var lineNo=1; lineNo <= pj.getLineItemCount('deduction'); lineNo++)
        pj.removeLineItem('deduction', lineNo );
}
```

```

    nlapiSubmitRecord(pj);
}

```

## Payroll Batch



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID of this record is payrollbatch.

This record is available only when the Payroll, Payroll Service, and Enhanced Premier Payroll features are enabled at Setup > Company > Enable Features on the Employees subtab.

In the user interface, you can access this record at Transactions > Employees > Create Payroll. For help working with this record in the user interface, see the help topic [Creating a Payroll Batch with Enhanced Premier Payroll](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted and searched using SuiteScript. Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes

## Usage Notes

Be aware of the following:

- After a payroll batch is created it is submitted twice. Once to calculate and again to commit.
- To calculate a payroll batch, it needs to be reloaded after it is submitted until one of the following 'status' is generated: calculated or error, committed or error.
- If you are calculating and committing a large amount of data a loop should be written to recheck the status of the payroll batch. Refer to the following table for more details.

Payroll Batch Status	Value	Description
Created	A	A new payroll batch has been created.
Calculated	B	A payroll batch has been calculated.
Edited	C	A payroll batch may have been calculated previously, or it is a newly created batch with some paychecks added or calculated.
About to Commit	D	Payroll is about to be committed.
Committed_At_Service	E	Payroll is committed for processing by payroll service, but some records need to be created in NetSuite.
Committed	F	Payroll is committed.
Completed	P	Payroll batch is completed.
Reversed	R	Payroll batch is reversed.
Error	X	Error occurred in payroll batch.

## Code Samples

The following code snippets show how to create a payroll batch and perform other basic tasks. It also includes how to add an employee to a payroll batch. For more information, see [Payroll Batch Employee](#).

```
<code>

//Create the payroll batch record
var record = nlapiCreateRecord('payrollbatch', 'true');
record.setFieldValue('offcycle', 'F');
record.setFieldValue('payfrequency', '52');
record.setFieldValue('periodending', '6/15/2016');

var id = nlapiSubmitRecord(record);

//Add employees to the payroll batch
//In most cases, you would iterate through the addmorepayeesmachine to find the employees you w
ould add to the batch
var addemp = nlapiLoadRecord('payrollbatchaddemployees', id, 'true');
addemp.setLineItemValue('addmorepayeesmachine','payemp', 1,'T');
nlapiSubmitRecord(addemp);

//Submit the record to be calculated
record=nlapiLoadRecord('payrollbatch', id, 'false');
nlapiSubmitRecord(record);

//Check the status of the record to ensure that calculating is complete before proceeding
do{
    record = nlapiLoadRecord('payrollbatch', id, 'false');
    status = record.getFieldValue('status');
}while ( !(status =='B' || status == 'E' || status =='X') );
```

```
//Set the flag to commit the payroll batch
if (statusa =='B') {
    record.setFieldValue('commit', 'T');

//Commit
nlapiSubmitRecord(record);

do {
    record = nlapiLoadRecord('payrollbatch', id, 'false');
    status = record.getFieldValue('status');
}while ( !(status == 'F' || status == 'E' || status == 'X') );

</code>
```

## Payroll Batch Employee

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This record enables you to add employees to the payroll batch.

The internal ID of this record is **payrollbatchaddemployees**.

This record is available only when the Payroll, Payroll Service, and Enhanced Premier Payroll features are enabled at Setup > Company > Enable Features on the Employees subtab.

In the user interface, you can access this record at Transactions > Employees > Create Payroll. For help working with this record in the user interface, see the help topic [Adding Employees to a Payroll Batch](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

## Supported Functions

The payroll batch employee record is read and edit only.

Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	no
nlapiCreateRecord	no
nlapiLoadRecord	yes
nlapiDeleteRecord	no
nlapiSearchRecord	no
nlapiSubmitRecord	yes

## Usage Notes

Be aware of the following:

- You cannot deselect an employee through SuiteScript once they have been added to the payroll batch and the record is submitted. If you need to remove an employee from the payroll batch you will need to do this through the UI.

## Purchase Contract



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This record enables you to take advantage of contracted quantity-based terms and discounts when creating purchase orders.

The internal ID of this record is `purchasecontract`.

This record is available only when the Purchase Contracts feature is enabled at Setup > Company > Enable Features, on the Transactions subtab.

In the user interface, you access this record at Transactions > Purchases > Enter Purchase Contracts. For help working with this record in the user interface, see the help topic [Creating Purchase Contracts](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the following table for more details.

Function	Supported?
<code>nlapiCopyRecord</code>	Yes
<code>nlapiCreateRecord</code>	Yes
<code>nlapiDeleteRecord</code>	Yes
<code>nlapiLoadRecord</code>	Yes
<code>nlapiSearchRecord</code>	Yes
<code>nlapiTransformRecord</code>	No

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Purchase Order



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `purchaseorder`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

This record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Requisition



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the requisition record to initiate the purchase process for goods and services needed within your company.

The internal ID for this record is `purchaserequisition`.

This record is available only when the Requisitions feature is enabled, at Setup > Enable Features, on the Transactions subtab. In the user interface, you access this record at Transactions > Purchases/Vendors > Enter Requisitions. For help working with this record in the user interface, see the help topic [Entering a Requisition](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the table below for more details.



Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	Yes, requisition records can be transformed into purchase order records.

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Samples

The following code snippets show how to add a requisition record and perform other basic tasks.

```
// Requisition SuiteScript name
var recType = 'PurchaseRequisition';

// Create and Add Record
var rec = nlapiCreateRecord(recType);
rec.setFieldValue('location',1);
rec.setLineItemValue('expense','account',1,59);
rec.setLineItemValue('expense','amount',1,2.3);
var recId = nlapiSubmitRecord(rec);

// Load and Update
rec = nlapiLoadRecord(recType,recId);
rec.setFieldValue('memo','memo');
nlapiSubmitRecord(rec);

// Search
var searchFilters = new Array();
searchFilters[0] = new nlobjSearchFilter('internalId', null, 'anyOf', recId);
var searchColumns = new Array();
searchColumns[0] = new nlobjSearchColumn('memo');
var result = nlapiSearchRecord('transaction', null, searchFilters, searchColumns);

// Copy
var rec2 = nlapiCopyRecord(recType, recId);
var rec2Id = nlapiSubmitRecord(rec2);

// Transformation (Initialization)
var po = nlapiTransformRecord(recType,rec2Id,"PurchaseOrder");
```

```

po.setFieldValue('entity','105');
var pold = nlapiSubmitRecord(po);

// Delete
nlapiDeleteRecord('PurchaseOrder',pold);
nlapiDeleteRecord(recType,recId);
nlapiDeleteRecord(recType,rec2Id);

```

## Return Authorization

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is returnauthorization.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

This record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Revenue Arrangement

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is revenuearrangement.

Revenue Arrangement is a transaction record that contains the details of customer performance obligations for purposes of revenue allocation and recognition.

This record is part of advanced revenue management. To use advanced revenue management, the Accounting Periods feature and Advanced Revenue Management feature must be enabled. Before you begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

The following table lists the scriptable body fields associated with this record.

Field	Type	Internal ID
Compliant	Check Box	compliant
Date	Date	trandate



Memo	Text Area	memo
Revenue Arrangement #	Text Area	tranid
Total Carve-out	Currency	totalallocationamount
Total Revenue Amount	Currency	totalrevenueamount
Transaction Is Allocation Bundle	Check Box	transvsoebundle
Transaction Total	Currency	totalsellingamount

## Revenue Element Sublist

The internal id for this sublist is `revenueelement`. The sublist type is inline editor.

The following table lists the scriptable line item fields on the Revenue Element sublist. Revenue elements correspond to individual lines in a source transaction.

Field	Type	Internal ID
Allocation Type	List/Record	allocationtype
Amortization End Date	Date	amortizationenddate
Amortization Start Date	Date	amortizationstartdate
Base Fair Value	Decimal Number	fairvalue
Calculated Fair Value Amount	Decimal Number	calculatedamount
Contract Expense Account	List/Record	contractexpenseacct
Contract Expense Offset Account	List/Record	contractexpenseoffsetacct
Cost Amortization Amount	Decimal Number	costamortizationamount
Deferral Account	List/Record	deferralaccount
End Date	Date	rerecenddate
Fair Value Override	Check Box	fairvalueoverride
VSOE	Check Box	isvsoeprice
Permit Discount	Check Box	permitdiscount
Reference ID	Long Text	referenceid
Recognition Account	List/Record	recognitionaccount
Return of Element	List/Record	returnofelement
Revenue Allocation Group	List/Record	revenueallocationgroup
Revenue Amount	Decimal Number	allocationamount
Revenue Recognition Rule	List/Record	revenuerecognitionrule
Start Date	Date	revrecstartdate

For help working with this record in the user interface, see the help topic [Managing Revenue Arrangements](#).



## Revenue Commitment

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `revenuecommitment`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

You cannot create this record using the standard `nlapiCreateRecord(...)` function. To create a Revenue Commitment record, you must execute a Sales Order to Revenue Commitment transformation. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Revenue Commitment Reversal

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `revenuecommitmentreversal`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

You cannot create this record using the standard `nlapiCreateRecord(...)` function. To create a Revenue Commitment Reversal record, you must execute a Return Authorization to Revenue Commitment Reversal transformation. Note that the Return Authorization must be approved and received for the transform to work.

In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Sales Order

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `salesorder`.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Sales orders brought into NetSuite using the eBay Integration feature will trigger standard SuiteScript user events (for example beforeSubmit, afterSubmit).

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
estgrossprofit	Est. Gross Profit	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
estgrossprofitpercent	Est. Gross Profit Percent	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search. When this field appears on the sublist line level, this field is not scriptable.
isrecurringpayment	Recurring Payment	A value for this field is stored only if the value for paymentmethod is a credit card.
totalcostestimate	Est. Extended Cost	When this field is on the body of the form in edit mode, this field is scriptable and can be returned in a transaction search.

Also note that this record has available transforms. See the SuiteScript Records Browser for available transforms. In the NetSuite Help Center, see [nlapiTransformRecord\(type, id, transformType, transformValues\)](#) for examples on how to transform records.

## Statistical Journal Entry



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The statistical journal entry record lets you increase or reduce the balance of a statistical account.

This internal ID for this record is `statisticaljournalentry`.

To use this record, the Statistical Accounts feature must be enabled at Setup > Enable Features, on the Accounting subtab. Also, you must have already created at least one statistical account. In the UI, you access this record at Transactions > Financial > Make Statistical Journal Entries. For help working with this record in the user interface, see the help topic [Making Statistical Journal Entries](#).

## Supported Script Types

The statistical journal entry record is scriptable in both client and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the table below for more details.

Function	Supported?
nlapiCopyRecord	Yes
nlapiCreateRecord	Yes
nlapiDeleteRecord	Yes
nlapiLoadRecord	Yes
nlapiSearchRecord	Yes
nlapiTransformRecord	No

## Usage Notes

Be aware of the following:

- The unitlabel field is read-only. It is populated automatically when you set the unitstype field. (The unitlabel value is the unitstype's base unit value. This value is defined in the corresponding unit of measure record.)
- The unitstype body field cannot be updated after the record is created.
- The field labeled Amount in the user interface is called debit in SuiteScript.
- Every statistical journal entry record must have at least one line.
- All sublist lines must use the same unit of measure, which is defined by the unitstype body field.

## Code Samples

The following samples show you can script to the statistical journal entry record.

### Adding and Deleting

The following code snippets show how you can create the record, add lines, load the record, and delete it.

```
// Note: These samples use constants. They are defined like: var subsidiaryId = '1';
// Create Record
var statisticalJournal = nlapiCreateRecord('STATISTICALJOURNALENTRY');
```



```

statisticalJournal.setFieldValue('subsidiary', subsidiaryId);
statisticalJournal.setFieldValue('externalid', externalId);
statisticalJournal.setFieldValue('unitstype', unitsTypeId);
statisticalJournal.setFieldValue('unit', unitId);

// Add line to the record
statisticalJournal.setLineItemValue('line', 'account', 1, statisticalAccountId);
statisticalJournal.setLineItemValue('line', 'debit', 1, amount); // field "debit" has label "Amount" in UI
statisticalJournal.setLineItemValue('line', 'lineunit', 1, unitId);
statisticalJournal.setLineItemValue('line', 'memo', 1, memo);
statisticalJournal.setLineItemValue('line', 'class', 1, classId);
statisticalJournal.setLineItemValue('line', 'department', 1, departmentId);
statisticalJournal.setLineItemValue('line', 'location', 1, locationId);

// Add record
var recId = nlapiSubmitRecord(statisticalJournal);

// Load record
var statisticalJournalAddedRec = nlapiLoadRecord('STATISTICALJOURNALENTRY', recId);

// Delete record
nlapiDeleteRecord('STATISTICALJOURNALENTRY', recId);

```

## Updating the Subsidiary Field

The subsidiary field can be updated only in dynamic mode, as shown in the following sample.

```

var recType = 'StatisticalJournalEntry';

// 1st Load record in Dynamic Mode
var rec = nlapiLoadRecord(recType,'168', {recordmode: 'dynamic'});

// 2nd Remove all lines
for (i=1; i <= rec.getLineItemCount('line'); i++)
{
    rec.removeLineItem('line','1');
}

// 3rd Change subsidiary
rec.setFieldValue('subsidiary','6');

// 4th Add new line for changed subsidiary
// the new account must be available in the new subsidiary
rec.selectNewLineItem('line');
rec.setCurrentLineItemValue('line','account','286');
rec.setCurrentLineItemValue('line','debit','5');
rec.commitLineItem('line');

nlapiSubmitRecord(rec);

```

## Time

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is timebill.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Transaction Search

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is transaction. Note that the transaction record is a **search record only**. You cannot create or copy this record.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Search Filters and Search Columns</b>		
ccnumber	Credit Card #	To prevent users from accessing sensitive information such as password and credit card data, this field cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).
entity	Name	The search filter entity is synonymous for the search filter name. Either filter can be used when searching the value of the Name / ID field in the UI.

## Transfer Order

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is transferorder.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Vendor Bill

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is vendorbill.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
usertotal	Amount	This field is not available via search or lookup for any transactions.

## Using Landed Cost Fields

When you create a landed cost category, the associated field IDs for the first category are landedcostamount1 and landedcostsource1. If you create a second category, the IDs will be landedcostamount2 and landedcostsource2.

This pattern increments by one with each additional category. For example, the IDs for the next landed cost category will be landedcostamount3 and landedcostsource3, and so on.

## Using Bill Address

You can use the Bill Address field on the vendor bill. For more information about creating and accessing subrecords, see [Working with Subrecords in SuiteScript](#).

## Vendor Credit

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is vendorcredit.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
usertotal	Amount	This field is not available via search or lookup for any transactions.

## Vendor Payment

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is vendorpayment.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Vendor Return Authorization

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is vendorreturnauthorization.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
usertotal	Amount	This field is not available via search or lookup for any transactions.



## Work Order



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `workorder`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Work Order Close



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `workorderclose`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

To use this record you must have the following features enabled: Work Orders and Manufacturing Work in Process.

In the UI, this record is accessed by going to Transactions > Manufacturing > Close Work Order.

Note these additional details:

- You must use `nlapiTransformRecord` to create a new instance of this record. In this case `workorder` is the originating record type. For more details, see "Prerequisites for Creating a Record," below.
- Assembly item should have scrap account, WIP account, and WIP Cost Variance Account specified.
- `nlapiCreateRecord` and `nlapiCopyRecord` are not supported on this record.

## Prerequisites for Creating a Record

Before you can create a work order issue record, a work order record must already exist, and the work order must be configured to use WIP (the WIP box on the work order record must be selected). This is true regardless of whether you are creating the work order issue record using initialize and add, or add by itself. If you try to create a work order issue record referencing a work order that has not been configured to use WIP, the system generates an error reading in part, "One of the following problems exists: You have an invalid work order < work order ID >, the work order does not use WIP, or

the work order is already closed." You can create and modify work orders by choosing Transactions > Manufacturing > Enter Work Orders.

You can also interact with work orders using SuiteScript, as described in [Work Order](#).

Note also that the assembly item referenced in the work order must be properly set up for WIP, as described in the [Setting Up Items as WIP Assemblies](#).

## Work Order Completion

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `workordercompletion`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

To use this record you must have the following features enabled: Work Orders and Manufacturing Work in Process.

In the UI, this record is accessed by going to Transactions > Manufacturing > Enter Completions.

Note these additional details:

- You must use `nlapiTransformRecord` to create a new instance of this record. In this case `workorder` is the originating record type. For more details, see "Prerequisites for Creating a Record," below.
- Assembly item should have scrap account and WIP account specified.
- `nlapiCreateRecord` and `nlapiCopyRecord` are not supported on this record.
- The Component sublist is available only when `backflush = true`

## Prerequisites for Creating a Record

Before you can create a work order issue record, a work order record must already exist, and the work order must be configured to use WIP (the WIP box on the work order record must be selected). This is true regardless of whether you are creating the work order issue record using initialize and add, or add by itself. If you try to create a work order issue record referencing a work order that has not been configured to use WIP, the system generates an error reading in part, "One of the following problems exists: You have an invalid work order < work order ID >, the work order does not use WIP, or the work order is already closed." You can create and modify work orders by choosing Transactions > Manufacturing > Enter Work Orders.

You can also interact with work orders using SuiteScript, as described in [Work Order](#).

Note also that the assembly item referenced in the work order must be properly set up for WIP, as described in the [Setting Up Items as WIP Assemblies](#).



# Work Order Issue

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `workorderissue`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

To use this record you must have the following features enabled: Work Orders and Manufacturing Work in Process.

In the UI, this record is accessed by going to Transaction > Manufacturing > Issue Components.

Note these additional details:

- You must use `nlapiTransformRecord` to create a new instance of this record. In this case `workorder` is the originating record type. For more details, see "Prerequisites for Creating a Record," below.
- `nlapiCreateRecord` and `nlapiCopyRecord` are not supported on this record.

## Prerequisites for Creating a Record

Before you can create a work order issue record, a work order record must already exist, and the work order must be configured to use WIP (the WIP box on the work order record must be selected). This is true regardless of whether you are creating the work order issue record using initialize and add, or add by itself. If you try to create a work order issue record referencing a work order that has not been configured to use WIP, the system generates an error reading in part, "One of the following problems exists: You have an invalid work order < work order ID >, the work order does not use WIP, or the work order is already closed." You can create and modify work orders by choosing Transactions > Manufacturing > Enter Work Orders.

You can also interact with work orders using SuiteScript, as described in [Work Order](#).

Note also that the assembly item referenced in the work order must be properly set up for WIP, as described in [Setting Up Items as WIP Assemblies](#).



# Support



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The following records are scriptable in SuiteScript:

- Case
- Issue
- Solution
- Task
- Topic

## Case



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `supportcase`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Change to Case Editing as of Version 2013 Release 2

Prior to Version 2013 Release 2, two users could edit the same case simultaneously and then both could save their changes. As of Version 2013 Release 2, case records have the same restrictions as other records. If a script is editing a case, and another user or script edits and saves the same case record, the script is unable to save the record. Any existing SuiteScript that updates cases even when they are open for editing by another user may need to be updated to continue to operate as intended.

## Issue



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `issue`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Solution



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `solution`.

This record is available when the Knowledge Base feature is enabled at Setup > Company > Enable Features, on the CRM tab. When the feature is enabled, you can access the inventory cost revaluation record in the UI by choosing Lists > Support > Solutions > New.

For details about working with this record manually, see the help topic [Creating Knowledge Base Solutions](#).

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Task



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is *task*.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Topic



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is *topic*.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

# File Cabinet

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The following records are scriptable in SuiteScript:

- Folder

## Folder

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is folder.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.



# Lists



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following list records are scriptable in SuiteScript:

- Account
- Accounting Book
- Accounting Context
- Accounting Period
- Allocation Schedule
- Amortization Schedule
- Amortization Template
- Billing Class
- Billing Rate Card
- Billing Schedule
- Bin
- Class
- Consolidated Exchange Rate
- Currency
- Customer Category
- Department
- Fair Value Price
- Gift Certificate
- Global Account Mapping
- Group
- Intercompany Allocation Schedule
- Inventory Number
- Item Account Mapping
- Item Revision
- Location
- Manufacturing Cost Template
- Manufacturing Routing
- Nexus
- Payroll Item
- Price Level
- Project Expense Type
- Revenue Recognition Plan

- Revenue Recognition Schedule
- Revenue Recognition Template
- Role
- Sales Tax Item
- Subsidiary
- Tax Control Account
- Tax Group
- Tax Period
- Tax Type
- Term
- Unit of Measure
- Vendor Category
- Workplace

## Account



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is account.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Accounting Book



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

When the Multi-Book Accounting feature has been enabled, you can use the accounting book record to create secondary books.

The internal ID for this record is accountingbook.

In the user interface, you access the accounting book record at [Setup > Accounting > Accounting Books > New](#).

## Supported Script Types

The accounting book record is supported in server SuiteScript only.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

This record is partially scriptable — it can be created, updated, deleted, and searched using SuiteScript.

For more details, refer to the table below.

Function	Supported?
nlapiCopyRecord	no
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create accounting book records and perform other basic tasks.

```
// Create Record

var rec = nlapiCreateRecord('accountingbook', null);
rec.setFieldValue('name', 'test');

var subs = new Array();
subs[0] = '1';
subs[1] = '3';
subs[2] = '5';

rec.setFieldValues('subsidiary', subs);
rec.setFieldValue('isprimary', 'T');

var recCreated = nlapiSubmitRecord(rec);

var id = 0;

// Update Record
var rec = nlapiLoadRecord('accountingbook', '3');

var subs = new Array();
subs[0] = '1';
subs[1] = '3';
subs[2] = '5';

rec.setFieldValues('subsidiary', subs);
rec.setFieldValue('isprimary', 'T');
```

```

var recUpdated = nlapiSubmitRecord(rec);

var id = 0;

// Delete Record
var rec = nlapiDeleteRecord('accountingbook', '2');

```

## Accounting Context



**Note:** The content in this help topic pertains to all versions of SuiteScript.

An accounting context can be a one-to-one relationship between a country's local GAAP (Generally Accepted Accounting Principles) reporting requirements and a statutory chart of accounts (COA). It can also be a unique relationship that meets your company's specific needs. Accounting contexts are useful when users prefer to work in a local GAAP context, rather than in the consolidated context with one centralized COA. Accounting contexts are also useful if you have Multi-Book Accounting provisioned in your account. You can set an accounting context specific for your secondary book and use it for your secondary book reports. For more information about accounting contexts, see the help topic [Setting Up Accounting Contexts](#).

The internal ID for this record is accountingcontext.

The accounting context record is available in OneWorld accounts.

The script user must have the Setup Company permission with Full permission level.

In the UI, you configure accounting contexts on the Accounting Contexts subtab on the General Preferences page at Setup > Company > General Preferences.. If you do not have a OneWorld account, this subtab is not visible.

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

## Supported Operations

This record is not fully scriptable. Supported operations are described in the following table.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes

Function	Supported?
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create an accounting context.

```
// SS 1.0
var rec = nlapiCreateRecord("accountingcontext");
rec.setFieldValue("name", "Context 1");
nlapiSubmitRecord(rec);
});
```

```
// SS 2.0
require(["N/record"], function(record){
  var rec = record.create({type: "accountingcontext"});
  rec.setValue({fieldId: "name", value: "Context 1"});
  rec.save();
});
```

## Accounting Period

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

When the Accounting Periods feature has been enabled, this record is available to SuiteScript.

In the user interface, you access accounting periods at Setup > Accounting > Manage Accounting Periods.

The internal ID for this record is accountingperiod.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Supported Script Types

The accounting period record is supported in client and server SuiteScript, for all script types.

## Supported Functions

Only search and read functions are supported.

## Code Sample

```
var period = nlapiLoadRecord('accountingperiod', <internal ID>);
var startDate = period.getFieldValue('startdate');
var endDate = period.getFieldValue('enddate');
```

## Allocation Schedule



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Allocation schedules transfer balances from expense accounts into one or more other accounts.

The internal ID for this record is **allocationschedule**.

The allocation schedule record is available only when the Accounting Periods and Expense Allocation features are enabled. An administrator can enable these features at Setup > Company > EnableFeatures > Accounting subtab. For information about expense allocation, see the help topic [Understanding Expense Allocation](#). For details on working with the allocation schedule in the user interface, see the help topic [Creating Expense Allocation Schedules](#).

If the Statistical Accounting feature is enabled, the allocation schedule record includes fields that enable you to base the weight for the allocation on the balance of a statistical account through statistical journals or as an absolute value. If the Dynamic Allocation feature is also enabled, the weight is dynamically calculated when the allocation journal is generated. An administrator can enable these features at Setup > Company > EnableFeatures > Accounting under Advanced Features. For information about statistical allocation schedules, see the help topic [Working with Allocation Schedules Weighted by the Balance of a Statistical Account](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

If the Dynamic Allocation feature is enabled, the currency field is not scriptable. None of the fields from the History subtab on the allocation schedule record in the user interface are not scriptable.

## Supported Operations

This record is not fully scriptable. Supported operations are described in the following table.

Function	Supported?
nlapiCopyRecord	no
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	no – Search is not supported because there is no search type AllocationSchedule.

Function	Supported?
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create a statistical allocation schedule that divides the balance of one account and transfers the amounts into two other accounts.

```
var record = nlapiCreateRecord("allocationschedule", false)

record.setFieldValue("name", "Allocation Schedule A");
record.setFieldValue("subsidiary", "1");

record.setLineItemValue("allocationsource", "account", 1, "24");

record.setLineItemValue("allocationdestination", "account", 1, "6");
record.setLineItemValue("allocationdestination", "weight", 1, "50.00");

record.setLineItemValue("allocationdestination", "account", 1, "32");
record.setLineItemValue("allocationdestination", "weight", 1, "50.00");

recordId = nlapiSubmitRecord(record);
```

## Amortization Schedule



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Amortization schedules are automatically created by the system for transactions that contain item or expense lines associated with amortization templates.

The internal ID for this record is `amortizationschedule`.

The amortization schedule record is available only when the Amortization feature is enabled, at Setup > Enable Features, on the Accounting subtab. In the user interface, you access this record at Lists > Accounting > Amortization Schedules. For details on working with this record in the user interface, see the help topic [Working with Amortization Schedules](#).

## Supported Script Types

This record is scriptable in server SuiteScript only.

The user events are not supported.

## Supported Functions

This record is not fully scriptable. Supported operations are described in the following table.

Function	Supported?
nlapiCopyRecord	no
nlapiCreateRecord	no
nlapiLoadRecord	yes
nlapiDeleteRecord	no — however, if you delete the parent record, the amortization schedule is also deleted.
nlapiSearchRecord	yes
nlapiSubmitRecord	yes — but note that edits are allowed only when the Allow Users to Modify Amortization Schedules preference has been selected. For details, see the help topic <a href="#">Setting Amortization Preferences</a> .
nlapiTransformRecord	no

## Usage Notes

Be aware of the following:

- As in the user interface, you cannot use external ID as part of your criteria when searching for amortization schedule records. You also cannot include external ID in search columns.
- The amortization schedule record is similar to the revenue recognition schedule record — so if you have existing integrations for revenue recognition schedules, you may be able to reuse elements of these scripts. The revenue recognition schedule record is described in [Revenue Recognition Schedule](#).

## Amortization Template



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the amortization template record to define the terms of amortization schedules, which are automatically created by the system. For example, on certain transactions, you can associate an amortization template with a line in the item or expense sublist. When the transaction is saved, an amortization schedule based on the template is automatically created. You can use amortization templates in conjunction with journal entry, vendor bill, and vendor credit records.

The internal ID for this record is `amortizationtemplate`.

The amortization template record is available only when the Amortization feature is enabled, at Setup > Enable Features, on the Accounting subtab. In the user interface, you access this record at Lists > Accounting > Amortization Templates > New. For help working with amortization templates in the user interface, see the help topic [Creating Amortization Templates](#).

## Supported Script Types

The amortization template record is supported in server SuiteScript only.

The user events are not supported.

## Supported Operations

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript.

Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Usage Notes

Be aware of the following:

- If the Method field (recurrencetype) is set to Custom, then you must include at least one line in the recurrence sublist. In this case, the template cannot be deleted.
- As in the user interface, you cannot use external ID as part of your criteria when searching for amortization template records. You also cannot include external ID in search columns.
- The amortization template record is similar to the revenue recognition template record — so if you have existing integrations for revenue recognition templates, you may be able to reuse elements of these scripts. The revenue recognition template record is described in [Revenue Recognition Template](#).

## Code Samples

The following code snippets show how to create amortization templates and perform other basic tasks.

```
// Create Record
var recAmTemp = nlapiCreateRecord('AmortizationTemplate');
recAmTemp.setFieldValue('name', 'Name');
recAmTemp.setFieldValue('externalid', 'externalId');
recAmTemp.setFieldValue('isamortization', 'T');
recAmTemp.setFieldValue('amortizationtype', 'STANDARD' ); // Type
recAmTemp.setFieldValue('recurrencetype', 'EVENPERIODSPRORATE' ); // Method
recAmTemp.setFieldValue('recogintervalsrc', 'RECEIPTDATE' ); // Term Source
recAmTemp.setFieldValue('revrecoffset', 0);
recAmTemp.setFieldValue('periodoffset', 1);
recAmTemp.setFieldValue('acctdeferral','1' );
recAmTemp.setFieldValue('acctcontra', '1');
recAmTemp.setFieldValue('accttarget', '1');
recAmTemp.setFieldValue('amortizationperiod',20);
```

```

recAmTemp.setFieldValue('residual','1.00' );
recAmTemp.setFieldValue('initialamount','12.00' );
recAmTemp.setFieldValue('isinactive', 'F');
var recId = nlapiSubmitRecord(recAmTemp);

// Load Record
var rec = nlapiLoadRecord('AmortizationTemplate',recId);

// Copy
var recCopied = nlapiCopyRecord('AmortizationTemplate', recId);

// Delete Record
nlapiDeleteRecord('AmortizationTemplate',recId);

```

## Billing Class

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You can use the billing class record to create different rates that you can use when calculating the cost of a resource's time on a project. You can use SuiteScript to create, update, and delete billing class records.

The internal ID for this record is `billingclass`.

The billing class record is available when the Per-Employee Billing Rates feature is enabled at Setup > Company > Enable Features, on the Employees tab. When the feature is enabled, you can access the billing class record in the UI by choosing Setup > Accounting > Billing Classes > New.

## Supported Script Types

The billing class record is scriptable in server SuiteScript only.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
<code>nlapiCopyRecord</code>	yes
<code>nlapiCreateRecord</code>	yes
<code>nlapiDeleteRecord</code>	yes
<code>nlapiSearchRecord</code>	yes
<code>nlapiSubmitRecord</code>	yes
<code>nlapiTransformRecord</code>	no

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Samples

The following sample shows how to create a billing class record.

```
var record = nlapiCreateRecord('billingclass');
record.setFieldValue('name', 'Billing Class');
record.setFieldValue('description', 'Billing Class Description');
record.setFieldValue('isinactive', 'F');
record.setLineItemValue('pricecost', 'price', 1, 2.56);
```

The following sample shows how to delete a billing class record.

```
nlapiDeleteRecord('billingclass', recId);
```

## Billing Rate Card



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `billingratecard`.

If you use billing classes, you can also use billing rate cards to define different billing rates for groups of billing classes. These rate cards can then be used to set billing rates on charge-based projects using time-based charge rules. You can use SuiteScript to read, create, update, delete, and search billing rate card records.

The billing rate card record is available when the Billing Rate Cards feature is enabled at Setup > Company > Enable Features, on the Employees tab. The Per-Employee Billing Rates and Charge-Based Billing features are also required to use billing rate cards. When the feature is enabled, you can access the billing rate card record in the UI by choosing Setup > Accounting > Billing Rate Cards > New.

## Supported Script Types

The billing rate card record is supported in client and server SuiteScript.

## Usage Notes

There are two sublists for rate card pricing: `ratecardpricing` and `ratecardpricingmulti`. These two lists are mutually exclusive. The first list, `ratecardpricing`, is a simple sublist used when the Multiple Currencies feature is not enabled. The second list, `ratecardpricingmulti`, is a matrix sublist used when the Multiple Currencies feature is enabled.

When this feature is enabled, the content of ratecardpricing is a subset of ratecardpricingmulti. If you enter information into ratecardpricing and then enable the Multiple Currencies feature, the original content is in a column in the ratecardpricingmulti sublist.

## Supported Functions

The following SuiteScript actions are supported for this record:

- Read
- Create
- Edit
- Delete
- Search

 **Note:** Copy and transform are not supported.

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Code Samples

The following sample shows a client script to create a new billing rate card with a single billing class without multiple currencies.

```
b = nlapiCreateRecord('billingratecard')
b.setFieldValue('name', 'abcd');
b.setLineItemValue('ratecardpricing', 'price', 1, 11);
nlapiSubmitRecord(b);
```

The following sample shows a client script to update the billing rate card name and a price in British pounds when multiple currencies is enabled.

```
b = nlapiLoadRecord('billingratecard', 3);
b.setFieldValue('name', b.getFieldValue('name') + ' - updated');
b.setLineItemValue('ratecardpricingmulti', 'price_2_', 1, 5);
nlapiSubmitRecord(b);
```

The following sample shows a user event script to update a price.

```
var brc = nlapiGetNewRecord();

var origPrice = brc.getLineItemValue('ratecardpricing', 'price', 1);
var newPrice = 11;
brc.setLineItemValue('ratecardpricing', 'price', 1, newPrice);
nlapiLogExecution('DEBUG', 'price changed from ' + origPrice + ' to ' + newPrice);
}
```

# Billing Schedule

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This record enables you to create a billing schedule that can be applied to a sales order, a line item on a sales order, or a project. SuiteScript supports all five types of billing schedules (charge-based, fixed bid interval, fixed bid milestone, standard, and time and materials).

The internal ID for this record is `billingschedule`.

To use the billing schedule record, you must enable the Advanced Billing feature, at Setup > Enable Features, on the Transactions subtab. In the UI, you access this record at Lists > Accounting > Billing Schedules > New. You can access a billing schedule of fixed bid milestone type through the project record's Financial tab.

## Supported Script Types

Billing Schedule is scriptable in server SuiteScript only.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the table below for more details.

Function	Supported?
<code>nlapiCopyRecord</code>	yes
<code>nlapiCreateRecord</code>	yes
<code>nlapiDeleteRecord</code>	yes
<code>nlapiSearchRecord</code>	yes
<code>nlapiSubmitRecord</code>	yes
<code>nlapiTransformRecord</code>	no

## Usage Notes

Additionally, note the following:

- To set a value for schedule type, which is a required body field, you must use `initializeValues`, not `setFieldValue`. For examples, see the following section.
- If you choose a schedule type of fixed bid milestone, you must identify an existing project record (or job record). You do so using `initializeValues`. Then, to create a link between the project and the new billing schedule, you must update the project record — this relationship is not established automatically when you create the billing schedule.
- The Recurrence sublist is available only when schedule type is set to Standard and frequency to Custom.
- The Milestone sublist is available only when schedule type is set to Fixed Bid Milestone.



## Code Samples

The following examples show how to create different types of billing schedules.

### Creating a Charge-Based Billing Schedule

The following sample shows how to create a charge-based billing schedule.

```
var RECORD_TYPE = 'billingschedule';
var SCHEDULE_TYPE = 'CB'; //Charge-Based

var recId = null;
var recName = SCHEDULE_TYPE + "record";

var initValues = new Array();
initValues.schedtype = SCHEDULE_TYPE;

//Create record
var bs = nlapiCreateRecord(RECORD_TYPE, initValues);

bs.setFieldValue('externalid', 'EXTID001');
bs.setFieldValue('name', recName);
bs.setFieldValue('frequency', 'DAILY');
bs.setFieldValue('dayperiod', '3');
recId = nlapiSubmitRecord(bs);
```

### Creating a Fixed Bid Milestone Billing Schedule

The following sample shows how to create a billing schedule of fixed bid milestone type. Note that this sample references a particular project record during the creation of the billing schedule. However, you still have to establish the relationship between the billing schedule and the project as a separate step.

```
var RECORD_TYPE = 'billingschedule';
var SCHEDULE_TYPE = 'FBM'; //Fixed Bid, Milestone
var P1 = '117'; //Project1
var P1M1 = '112'; //Project1 - Milestone1
var P1M2 = '113'; //Project2 - Milestone2

var recId = null;
var recName = SCHEDULE_TYPE + " record";

var initValues = new Array();
initValues.schedtype = SCHEDULE_TYPE;
initValues.project = P1;

//Create the record
var bs = nlapiCreateRecord(RECORD_TYPE, initValues);

bs.setFieldValue('name', recName);
bs.setFieldValue('initialamount', '10%');

//Create the sublist
```

```

bs.selectNewLineItem('milestone');
bs.setCurrentLineItemValue('milestone', 'milestoneamount', '25%');
bs.setCurrentLineItemValue('milestone', 'milestonedate', '11/21/2013');
bs.setCurrentLineItemValue('milestone', 'projecttask', P1M1);
bs.commitLineItem('milestone');

//Update the project (link it with the newly created Billing Schedule)
var project = nlapiLoadRecord('job', P1);
project.setFieldValue('jobbillingtype', SCHEDULE_TYPE);
project.setFieldValue('billingschedule', recId);
var pld = nlapiSubmitRecord(project);

```

## Bin

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is bin.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

## Class

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is classification.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Consolidated Exchange Rate

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

If you use NetSuite OneWorld and your subsidiaries have different base currencies, you can maintain a table of consolidated exchange rates. This table is used to ensure that for consolidation purposes, currency amounts properly roll up from child to parent subsidiaries. Each consolidated exchange



rate translates between the base currency of a subsidiary and the base currency of its parent or grandparent subsidiary, for a specified accounting period. Consolidated exchange rates include three different rate types per period and subsidiary pair: Current, Average, and Historical.

The internal ID for this record is `consolidatedexchangerate`.

The consolidated exchange rate record is available only when the Multiple Currencies feature is enabled. An administrator can enable this feature at Setup > Company > EnableFeatures > Company subtab. For information about consolidated exchange rates, see the help topic [Using Consolidated Exchange Rates](#).

The script user must have the Currency permission with Full permission level. To edit the different rate types per period and subsidiary pair, the script user must also have access to both the From and To subsidiary.

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

The editable elements of the record are the `currentrate`, `averagerate`, `historicalrate`, and `externalid`. The remaining record elements can be read and searched.



**Important:** Rates are created and deleted when subsidiaries or secondary accounting books are created and deleted. Script users can edit the record only if it is not derived (Element field-> `isderived`) and only when the accounting period to which this rate belongs is not closed (Element field-> `isperiodclosed`). In addition, the script user cannot edit a consolidated exchange rate of 1 for an elimination subsidiary and its direct parent (Element field-> `iseliminationsubsidiary`), with the following exception. Those customers who already have a non-1 consolidated exchange rate (current, average, or historical) between an elimination subsidiary and its direct parent subsidiary can edit the consolidated exchange rates for the elimination subsidiary.

## Supported Operations

This record is not fully scriptable. Supported operations are described in the following table.

Function	Supported?
<code>nlapiCopyRecord</code>	no
<code>nlapiCreateRecord</code>	no
<code>nlapiLoadRecord</code>	yes
<code>nlapiDeleteRecord</code>	no
<code>nlapiSearchRecord</code>	yes
<code>nlapiSubmitRecord</code>	yes
<code>nlapiTransformRecord</code>	no

## Code Samples

The following code snippets show how to update a consolidated rate between two subsidiaries.

```

var PARENT_SUBSIDIARY = 1;
var CHILD_SUBSIDIARY = 3;
var PERIOD = 212;

// Search for the consolidated rate record from CHILD_SUBSIDIARY to PARENT_SUBSIDIARY for a given PERIOD
var filters = new Array();
var columns = new Array();
filters[0] = new nlobjSearchFilter('fromsubsidiary', null, 'is', CHILD_SUBSIDIARY);
filters[1] = new nlobjSearchFilter('tosubsidiary', null, 'is', PARENT_SUBSIDIARY);
filters[2] = new nlobjSearchFilter('period', null, 'is', PERIOD);
columns[0] = new nlobjSearchColumn('internalid');
var results = nlapiSearchRecord('consolidatedexchangerate', null, filters, columns);

// Update the consolidated rate's current, average, and historical rate
var rateRecord = nlapiLoadRecord('consolidatedexchangerate', results[0].getValue('internalid'), true);
rateRecord.setFieldValue('currentrate', 1.2);
rateRecord.setFieldValue('averagerate', 1.3);
rateRecord.setFieldValue('historicalrate', 1.4);
nlapiSubmitRecord(rateRecord);

```

## Currency

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is currency.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

When the Multiple Currencies feature is enabled, full server-side scripting is supported for this record type.

When the Multiple Currencies feature is not enabled, scripting does not support create, edit, delete, or search of currency records. Search-based functions such as nlapiLookupField are not supported, because search is not supported. Loading of a currency record to get field values is supported, as long as the currency ID is known, as shown in the following example:

```

var rec = nlapiLoadRecord('currency', 1);
var symbol=rec.getFieldValue('symbol');
var a = 1;

```

As of Version 2016 Release 1, the currencyprecision field is available for scripting, even when the ALLOWCURRENCYPRECISIONCHANGE preference is disabled and the field is read-only. In scripting, this field is always a number and never a list box, behavior which differs from this field's behavior in the UI.

## Customer Category

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `customercategory`. Search is not available on this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Department

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `department`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Expense Category

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `expensecategory`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Fair Value Price

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `fairvaluepricelist`.

Fair Value Price is a list of the records that define the fair value for items. Fair value price is used to allocate revenue in revenue arrangements

This record is part of advanced revenue management. To use advanced revenue management, the Accounting Periods feature and Advanced Revenue Management feature must be enabled. Before you



begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

The following table lists the scriptable fields associated with this record.

Field	Type	Internal ID
Currency	List/Record	currency
End Date	Date	enddate
Fair Value	Decimal Number	fairvalue
Fair Value Formula	List/Record	fairvalueformula
Fair Value Range Checking	List/Record	fairvaluerangepolicy
High Value	Decimal Number	highvalue
High Value Percent	Percent	highvaluepercent
Low Value	Decimal Number	lowvalue
Low Value Percent	Percent	lowvaluepercent
Is VSOE Price?	Check Box	isvsoeprice
Item	List/Record	item
Item Revenue Category	List/Record	itemrevenuecategory
Start Date	Date	startdate

For help working with this record in the user interface, see the help topic [Setting Up Fair Value](#).

## Gift Certificate

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is *giftcertificate*.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Global Account Mapping

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

For accounts using Multi-Book Accounting, the global account mapping record enables you to configure secondary accounting books to post to accounts different from the primary book. These mappings are used by transactions where the user can manually select the account to which the transaction posts.

The internal ID for this record is *globalaccountmapping*.



Using this record requires that, as part of your Multi-Book configuration, you select the Chart of Accounts Mapping option at Setup > Enable Features, on the Accounting subtab, in addition to the other setup steps required for Multi-Book Accounting.

In the user interface, you access this record at Setup > Accounting > Global Account Mappings > New.

## Supported Script Types

This record is scriptable in server SuiteScript only.

All three user events are supported: before Load, before Submit, and afterSubmit.

## Supported Functions

This record is fully scriptable, which means it can be created, updated, copied, deleted, and searched using SuiteScript.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create global account mapping records and perform other basic tasks.

```
// Create Record
var rec = nlapiCreateRecord('globalaccountmapping', null);
rec.setFieldValue('effectivedate', '4/4/2004');
rec.setFieldValue('accountingbook', '2');
rec.setFieldValue('sourceaccount', '6');
rec.setFieldValue('subsidiary', '1');
rec.setFieldText('class', 'Class US');
rec.setFieldText('department', 'Department US');
rec.setFieldText('location', 'Location US');
rec.setFieldValue('destinationaccount', '6');
var id = nlapiSubmitRecord(rec);
var x = 0;

// Update Record
var rec = nlapiLoadRecord('globalaccountmapping', '102');
rec.setFieldValue('effectivedate', '5/5/2005');
```

```

rec.setFieldText('sourceaccount', 'Advances Paid');
rec.setFieldText('destinationaccount', 'ABN Withholding');

var id = nlapiSubmitRecord(rec);

// Delete Record
var rec = nlapiDeleteRecord('globalaccountmapping', '102');

// Copy Record
var rec = nlapiCopyRecord('globalaccountmapping', '103');
rec.setFieldValue('effectivedate', '5/5/2005');
rec.setFieldText('sourceaccount', 'Advances Paid');
rec.setFieldText('destinationaccount', 'ABN Withholding');
var id = nlapiSubmitRecord(rec);

```

## Group

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You use the group record to define groups of contacts, customers, employees, partners, or vendors.

The internal ID for this record is entitygroup.

In the UI, you access this record at Lists > Relationships > Group > New.

## Supported Script Types

The group record is scriptable in server SuiteScript only. None of the user events is supported.

## Supported Functions

This record is fully scriptable, which means it can be created, read, updated, deleted, and searched. Refer to the table below for more details.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiDeleteRecord	yes
nlapiLoadRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Usage Notes

Be aware of the following details when working with this record:

- The grouptype field is required and must be set either to "static" or "dynamic."
- If the grouptype is dynamic, you can never manually add to the Members sublist (because the group members are determined by a saved search). This behavior is the same as in the user interface.
- If the grouptype is static, you can create lines in the Members sublist during the time that you are creating the record, but you cannot modify the sublist during updates. This behavior differs from the user interface, which does allow manual updates after the record is created.

To add members to a group in SuiteScript after it has been created, you can use code like the following:

```
nlapiAttachRecord('customer', <someCustomerId>, 'entitygroup', <someGroupId>);
nlapiDetachRecord('customer', <someCustomerId>, 'entitygroup', <someGroupId>);
```

## Code Sample

The following sample shows how to create a static group record.

```
var initValues = new Array();
initValues.grouptype = 'Employee';
initValues.dynamic = 'F';
var wc1 = nlapiCreateRecord('entitygroup', initValues);
wc1.setFieldValue('groupname', 'WC1');
wc1.setFieldValue('subsidiary', '1');
wc1.setFieldValue('ismanufacturingworkcenter', 'T');
wc1.setFieldValue('machineresources', '11');
wc1.setFieldValue('laborresources', '22');
var wc1Id = nlapiSubmitRecord(wc1);
```

## Intercompany Allocation Schedule



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Intercompany allocation schedules transfer a balance from one source subsidiary account to multiple destination subsidiaries for costs that are shared between subsidiaries.

The internal ID for this record is intercompallocationschedule.

The intercompany allocation schedule record is available only in OneWorld accounts and when the Accounting Periods and Expense Allocation features are enabled. An administrator can enable these features at Setup > Company > EnableFeatures > Accounting subtab. For information about expense allocation, see the help topic [Understanding Expense Allocation](#). For details on working with the intercompany allocation schedule in the user interface, see the help topic [Creating Intercompany Allocation Schedules](#).

If the Statistical Accounting feature is enabled, the intercompany allocation schedule record includes fields that enable you to base the weight for the allocation on the balance of a statistical account

through statistical journals or as an absolute value. If the Dynamic Allocation feature is also enabled, the weight is dynamically calculated when the intercompany allocation journal is generated. An administrator can enable these features at Setup > Company > EnableFeatures > Accounting under Advanced Features. For information about statistical allocation schedules, see the help topic [Working with Allocation Schedules Weighted by the Balance of a Statistical Account](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

If the Dynamic Allocation feature is enabled, the currency field is not scriptable. None of the fields from the History subtab on the intercompany allocation schedule record in the user interface are not scriptable.

## Supported Operations

This record is not fully scriptable. Supported operations are described in the following table.

Function	Supported?
nlapiCopyRecord	no
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	no – Search is not supported because there is no search type AllocationSchedule.
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create a statistical intercompany allocation schedule that divides the balance of one account and transfers the amounts into two other subsidiaries.

```
var record = nlapiCreateRecord("intercompallocationschedule", false)

record.setFieldValue("name", "Allocation Schedule A");
record.setFieldValue("subsidiary", "1");

record.setLineItemValue("allocationsource", "account", 1, "24");

record.setLineItemValue("allocationdestination", "subsidiary", 1, "6");
record.setLineItemValue("allocationdestination", "weight", 1, "50.00");

record.setLineItemValue("allocationdestination", "subsidiary", 1, "32");
record.setLineItemValue("allocationdestination", "weight", 1, "50.00");
```

```
recordId = nlapiSubmitRecord(record);
```

## Inventory Number

**i Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is inventorynumber.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

Copy, Create, and Delete are not allowed for this record.

## Item Account Mapping

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

For accounts using Multi-Book Accounting, the item account mapping record enables you to configure secondary accounting books to post to accounts different from the primary book. These mappings are used by transactions where the item determines the account to which the transaction posts.

The internal ID for this record is itemaccountmapping.

Using this record requires that, as part of your Multi-Book configuration, you select the Chart of Accounts Mapping option at Setup > Enable Features, on the Accounting subtab, in addition to the other setup steps required for Multi-Book Accounting.

In the user interface, you access this record at Setup > Accounting > Item Account Mappings > New.

## Supported Script Types

This record is scriptable in server SuiteScript only.

All three user events are supported: before Load, before Submit, and afterSubmit.

## Supported Functions

This record is fully scriptable, which means that it can be created, updated, copied, deleted, and searched using SuiteScript. For more details, refer to the table below.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Code Samples

The following code snippets show how to create item account mapping records and perform other basic tasks.

```
// Get Record
var rec = nlapiLoadRecord('itemaccountmapping', '105');
var id = 0;

// Create Record
var rec = nlapiCreateRecord('itemaccountmapping', null);
rec.setFieldValue('effectivedate', '4/4/2004');
rec.setFieldValue('accountingbook', '2');
rec.setFieldValue('sourceaccount', '118');
rec.setFieldText('itemaccount', 'Asset');
rec.setFieldValue('subsidiary', '3');
rec.setFieldText('class', 'Class US');
rec.setFieldText('department', 'Department US');
rec.setFieldText('location', 'Location US');
rec.setFieldValue('destinationaccount', '118');
var id = nlapiSubmitRecord(rec);
var x = 0;

// Update Record
var rec = nlapiLoadRecord('itemaccountmapping', '201');
rec.setFieldValue('effectivedate', '5/5/2005');
rec.setFieldValue('subsidiary', '1');
rec.setFieldText('itemaccount', 'Discount');
rec.setFieldText('sourceaccount', 'Advertising');
rec.setFieldText('destinationaccount', 'Advertising');
var id = nlapiSubmitRecord(rec);

// Copy Record
var rec = nlapiCopyRecord('itemaccountmapping', '201');
rec.setFieldValue('effectivedate', '6/6/2005');
rec.setFieldText('itemaccount', 'Asset');
rec.setFieldValue('sourceaccount', '118');
```

```

rec.setFieldValue('destinationaccount', '118');
var id = nlapiSubmitRecord(rec);

// Delete Record
var rec = nlapiDeleteRecord('itemaccountmapping', '201');

```

## Item Revision



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `itemrevision`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

## Sample Code

The following example creates an item revision record:

```

function afterSubmit(type)
{
    var itemrev = nlapiCreateRecord("itemrevision");

    itemrev.setFieldValue("name", "revision name 222");
    itemrev.setFieldValue("item", "109");
    itemrev.setFieldValue("memo", "revision memo");
    itemrev.setFieldValue("effectivedate", "3/4/2012");

    var id = nlapiSubmitRecord(itemrev, true);
}

```

## Location

The internal ID for this record is `location`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Manufacturing Cost Template



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is manufacturingcosttemplate.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

### Usage Notes

To work with this record, the Manufacturing Routing and Work Center feature must be enabled at Setup > Company > Enable Features, on the Items & Inventory tab.

In the UI, this record is accessed by going to Lists > Supply Chain > Manufacturing Cost Template.

## Manufacturing Routing



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is manufacturingrouting.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

### Usage Notes

To work with this record, the Manufacturing Routing and Work Center feature must be enabled at Setup > Company > Enable Features, on the Items & Inventory tab.

In the UI, this record is accessed by going to Lists > Supply Chain > Manufacturing Routing.

## Nexus



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is nexus.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Payroll Item



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `payrollitem`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The exposure of this record to SuiteScript is intended to enable global payroll solutions. You can use it along with the [Paycheck Journal](#) record to create custom payroll solutions and to support integrations with external payroll systems.

## Code Samples

The following samples create Deduction type payroll items.

```
function createPayrollItemdeductionMinimal()
{
    var pi = nlapiCreateRecord('payrollitem');
    pi.setFieldValue('subsidiary', 1);
    pi.setFieldValue('itemtype', 16);
    pi.setFieldValue('liabilityaccount', 27);
    pi.setFieldValue('name', 'SSSitem-Deduction-Minimal');
    pi.setFieldValue('custrecord_payroll_item', 'Cust_field');
    pi.setFieldValue('externalid', 'testingexternalID');
    nlapiSubmitRecord(pi);
}

function createPayrollItemdeductionComplete()
{
    var pi = nlapiCreateRecord('payrollitem');
    pi.setFieldValue('externalid', 'SSSitem-Deduction-Completed');
    pi.setFieldValue('subsidiary', 1);
    pi.setFieldValue('itemtype', 16);
    pi.setFieldValue('liabilityaccount', 150);
    pi.setFieldValue('name', 'SSSitem-Deduction');
    pi.setFieldValue('vendor', 1);
    pi.setFieldValue('employeepaid', true);
    pi.setFieldValue('custrecord_payroll_item', 'Cust_field');
```

```

    nlapiSubmitRecord(pi);
}

```

The following samples create Earning:Addition type payroll items.

```

function createPayrollItemAdditionMinimal()
{
    var pi = nlapiCreateRecord('payrollitem');
    pi.setFieldValue('subsidiary', 1);
    pi.setFieldValue('itemtype', 6);
    pi.setFieldValue('expenseaccount', 114);
    pi.setFieldValue('name', 'SSSItem-addition-Minimal');
    pi.setFieldValue('custrecord_payroll_item', 'Cust_field');

    nlapiSubmitRecord(pi);
}

function createPayrollItemAdditionComplete()
{
    var pi = nlapiCreateRecord('payrollitem');
    pi.setFieldValue('externalid', 'SSSItem-Addition-Completed');
    pi.setFieldValue('subsidiary', 1);
    pi.setFieldValue('itemtype', 6);
    pi.setFieldValue('expenseaccount', 114);
    pi.setFieldValue('name', 'SSSItem-Addition');
    pi.setFieldValue('custrecord_payroll_item', 'Cust_field');
    nlapiSubmitRecord(pi);
}

```

The following code updates a payroll item.

```

function updatePayrollItem()
{
    var recordpi = nlapiLoadRecord('payrollitem', 110);

    recordpi.setFieldValue('liabilityaccount', 29);
    recordpi.setFieldValue('vendor', 6);
    recordpi.setFieldValue('custrecord_payroll_item', 'Updated');
    recordpi.setFieldValue('inactive', 'T');

    nlapiSubmitRecord(recordpi);
}

```

The following code deletes a payroll item.

```
function deletePayrollItem()
```

```

{
  var savedSearchInternalId = 2
  var searchresults = nlapiSearchRecord('payrollitem', savedSearchInternalId, null, nu
ll);
  for ( var i = 0; searchresults != null && i < searchresults.length; i++ )
  {
    var searchresult = searchresults[ i ];
    if ( 0 < searchresults[i].getId() )
      nlapiDeleteRecord(searchresults[i].getRecordType(), searchresults[i].getId());
  }
}

```

## Price Level



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is *pricelvel*. Search is not available on this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Project Expense Type



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.



**Important:** The project expense type record is part of the Job Costing and Project Budgeting feature. For information on the availability of this feature, please contact your account representative.

The project expense type record lets you create expense classifications that you can apply to projects. The internal ID for this record is *projectexpensetype*.

This record is available when the Job Costing and Project Budgeting feature is enabled at Setup > Company > Enable Features, on the Company tab. If you do not see this option in the UI, contact your account representative for assistance.

When the feature is enabled, you can access the project expense type record in the UI by choosing Setup > Accounting > Setup Tasks > Project Expense Types > New. For help manually creating a project expense type record, see the help topic [Creating Project Expense Types](#).

## Supported Script Types

This record is not scriptable in client SuiteScript. It is scriptable in server SuiteScript only.

## Supported Functions

This record is fully scriptable. Refer to the table below for more details.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes
nlapiTransformRecord	no

## Usage Notes

The Language sublist is available only when the Multi-Language feature is enabled at Setup > Company > Enable Features, on the Company tab. The sublist's records can be updated, but you cannot add records to the sublist through SuiteScript.

## Field Definitions

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Revenue Recognition Plan

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `revrecplan`.

This record is used to indicate the posting periods in which revenue should be recognized and the amount to be recognized in each period. Revenue plans are derived from revenue recognition rules. Each revenue element has a forecast plan and one or more actual plans. The actual revenue plans control the posting of revenue.

This record is part of advanced revenue management. To use advanced revenue management, the Accounting Periods feature and Advanced Revenue Management feature must be enabled. Before you begin working with advanced revenue management programmatically, see the help topic [Setting Up Advanced Revenue Management](#).

The following table lists the scriptable fields associated with this record.

Field	Type	Internal ID
Amount	Decimal Number	amount

Catch Up Period	List/Record	catchupperiod
Comments	Text Area	comments
Hold Revenue Recognition	Check Box	holdrevenuerecognition
Rev Rec End Date	Date	revrecenddate
Rev Rec Start Date	Date	revrecstartdate

## Planned Revenue Sublist

The internal id for this sublist is plannedrevenue. The sublist type is inline editor.

The following table lists the scriptable sublist fields associated with the Planned Revenue sublist.

Field	Type	Internal ID
Amount	Decimal Number	amount
Planned Period	List/Record	plannedperiod

For help working with this record in the user interface, see the help topic [Working with Revenue Recognition Plans](#).

## Revenue Recognition Schedule

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is revrecschedule.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only. Also, user event scripts are not supported.

Copy, Create, and Delete are not allowed for this record.

## Revenue Recognition Template

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is revrectemplate.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only. Also, user event scripts are not supported.

## Role

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is **role**.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Supported Functions

This record is not fully scriptable. Only search is permitted.

## Usage Notes

Client SuiteScript is not supported for this record. It is scriptable in server SuiteScript only.

To work with this record, a user must have Manage Roles permissions, at Setup > Users/Roles > Manage Roles > Search.

In SuiteScript 1.0, nlapiSearchRecord() must be called with **Role** as its first parameter:

```
var roles = nlapiSearchRecord('Role', SavedSearchToBeLoaded, filters, columns);
```

In SuiteScript 2.0, role is called as **Role=search.Type.ROLE**

Does not support the nlapiLoadSearch(type, id) search API.

## Sales Tax Item

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is **salestaxitem**.



See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Subsidiary

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is subsidiary.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The Company Information page and the root Subsidiary record share information. In OneWorld accounts, if you update shared fields on the Company Information page, the corresponding fields on the root Subsidiary record are also updated. An update to the Company Information page also triggers all user event scripts deployed on the Subsidiary record. If your script updates a shared field on the Subsidiary record, the corresponding field on the Company Information page is also updated.

The Subsidiary record is hidden in accounts that are not OneWorld, but the record still exists. You cannot deploy a stand-alone user event script on the Subsidiary record in accounts that are not OneWorld. You can, however, bundle your script and distribute it to these accounts. In this scenario, updates to the Company Information page trigger the execution of the bundled script.

## Tax Control Account

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

A tax control account is an account to which the amounts computed for indirect taxes, such as sales tax and VAT, are posted.

The internal ID for this record is taxacct.

In the UI, you can create a tax control account, and view existing ones, at Setup > Accounting > Tax Control Accounts. Note also that a tax control account is essentially an account record, with a few differences. Because they are accounts, tax control accounts also show up in the full list of accounts at Lists > Accounting > Accounts.

## Supported Script Types

This record is scriptable in both client and server SuiteScript. However, user event scripts are *not* supported.



## Supported Functions

This record is not fully scriptable. Only creating, copying, and some updates are permitted, as described below.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiDeleteRecord	no
nlapiLoadRecord	yes
nlapiSearchRecord	no
nlapiTransformRecord	no

## Updating

Updating is permitted, but only of certain fields: [Description](#), [External ID](#), [IsInactive](#), and [Name](#). However, you may be able to change other fields when interacting with the tax control account as an account. For details, see [Account](#).

## Alternatives to Deleting

nlapiDeleteRecord is not supported, because a tax control account cannot be deleted (either through the UI or through SuiteScript). If the Advanced Taxes feature is enabled, you can effectively remove the tax control account by deleting the nexus that the account is associated with. If the Advanced Taxes feature is *not* enabled, you cannot remove the account.

An alternative to deleting the account is to make it account inactive, by setting the [IsInactive](#) field to true.

For more details on accounts that cannot be deleted in NetSuite, refer to the help topic [Deleting Accounts and Making Accounts Inactive](#).

## Field Definitions

This section describes some of the key fields on the tax control account.

### Country

The value for country is derived from the nexus value and is read-only. If your account does not have the Advanced Taxes feature enabled, the value for country will always be the same for all your tax control accounts.

### Description

The description of the record can include only up to 50 characters. Otherwise, the operation fails with an error reading “The field description contained more than the maximum number ( 50 ) of characters allowed.”

Description is one of the few fields that can be modified during an update.

## External ID

ExternalID is one of the few fields that can be modified during an update.

## IsInactive

Because a tax control account cannot be deleted, you might want to make it inactive.

IsInactive is one of the few fields that can be modified during an update.

## Name

The name of the record must be unique. If you try to add a tax control account using a non-unique value for the name field, the system returns an error reading “This record already exists,” even if you included a unique external ID.

Name is one of the few fields that can be modified during an update.

## Nexus

You can set a value for nexus only if the Advanced Taxes feature is enabled. If your NetSuite account does not use the Advanced Taxes feature, the value for nexus is set automatically and will always be the same for all your tax control accounts.

When Advanced Taxes is enabled, initialization of a nexus value is required. (For an example, refer to [Code Samples](#).) However, after the account is created, the value for nexus cannot be changed.

Note that Advanced Taxes is enabled in all OneWorld accounts and cannot be turned off.

## Tax Account Type

All tax control accounts have a tax account type, but you only actively choose a tax account type for accounts in certain countries. For example:

- If you are creating a tax code for the United Kingdom, valid choices include “sales” and “purchase.”
- When creating a tax code for the United States, only one option exists (“sales”), so you do not set a value for this field.

In countries where both “sales” and “purchase” are valid choices, the tax account type field is required. After an account has been created, the tax account type cannot be changed.

In countries where only one choice is allowed, tax account type is automatically set, and you cannot change it through SuiteScript (or through the UI).

Note that if your account uses Advanced Taxes and has nexuses in many countries, you may need to set this field for some tax control accounts and not others.

## For More Information

See the SuiteScript Records Browser for all internal IDs associated with this record.

## Usage Notes

This section includes additional details on interacting with the tax control account record.

### Finding the Internal ID

If you need the internal ID for an existing tax control account, note that you cannot find it through Setup > Accounting > Tax Control Accounts. However, you can find it at Lists > Accounting > Accounts. Make sure you have configured your NetSuite preference to "Show Internal IDs." (You can find this choice at Home > Set Preferences.)

When sorting accounts at Lists > Accounting > Accounts, be aware that:

- Tax control accounts of type "sale" show up as Other Current Liability.
- Tax control accounts of type "purchase" show up as Other Current Asset.

Be aware that you can also add and update values for the external ID field.

### Relationship to the Account Record

You can interact with the tax control account in many of the same ways you can interact with any account. However, some exceptions exist. For example, a tax control account cannot be deleted, and there are limits to what you can update, as described in [Updating](#). However, when you interact with the record as an account, you may be able to update additional fields.

For details on interacting with the account record, see [Account](#).

## Code Samples

The following examples show how you can script with tax control account records.

### Get

The following sample shows how to retrieve a tax account record with the internal ID of "186":

```
var taxAcct = nlapiLoadRecord('taxacct', '186');
;
```

### Add

The following example shows how to create a tax control account when the Advanced Taxes feature is enabled. In this scenario, a value for nexus is required:

```
var taxAcct = nlapiCreateRecord('taxacct', {'nexus', 'internal ID of nexus'});
taxAcct.setFieldValue('name', 'Name of account');
taxAcct.setFieldValue('description', 'Description of account');
taxAcct.setFieldValue('externalid', 'Your external ID value');
taxAcct.setFieldValue('taxaccttype', 'Type');
```



```
nlapiSubmitRecord(taxAcct);
```

## Update

This example shows how to update the name, description, and external ID of a tax control account, and to make it active. Note that these fields are the only ones that can be updated.

```
var taxAcct = nlapiLoadRecord('taxacct', 'internal ID of tax control account');
taxAcct.setFieldValue('name', 'Updated name');
taxAcct.setFieldValue('description', 'Updated description');
taxAcct.setFieldValue('isinactive', 'F');
taxAcct.setFieldValue('externalid', 'Updated external ID of tax control account');
nlapiSubmitRecord(taxAcct);
;
```

## Tax Group



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

You can use the tax group record to combine several tax codes, even if the taxes are paid to different jurisdictions. For example, a tax group in the US might include a state tax, a city tax, and a transit tax. The advantage of using a tax group is that, when you create a sales invoice, you can apply one tax group to the transaction, instead of several separate tax codes.

The internal ID for the tax group record is `taxgroup`.

In the UI, you navigate to this record by choosing **Setup > Accounting > Tax Groups**.

## Supported Script Types

The tax group record is scriptable in both client SuiteScript and Server SuiteScript.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

This record is fully scriptable. Refer to the table below for details on supported API functions.

Function	Supported?
<code>nlapiCopyRecord</code>	yes
<code>nlapiCreateRecord</code>	yes
<code>nlapiDeleteRecord</code>	yes
<code>nlapiSearchRecord</code>	yes

Function	Supported?
nlapiTransformRecord	no

## Field Definitions

See the [SuiteScript Records Browser](#) for the internal IDs of fields, search filters, and search columns associated with this record.

For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).

## Code Samples

The following sample shows how to create a US tax group.

```
var initValues = new Array();
initValues.nexuscountry = 'US';
var tg = nlapiCreateRecord('taxgroup', initValues);
tg.setFieldValue('itemid', 'Test US Tax Group');
tg.setFieldValue('description', 'Tax group description');
tg.selectNewLineItem('taxitem');
tg.setCurrentLineItemValue('taxitem', 'taxname', '-1425'); // -1425 is ID of TX_BUFFALO Tax Cod
e
tg.commitLineItem('taxitem');
recId = nlapiSubmitRecord(tg);
;
```

The following code sample shows how to create a tax group for Canada.

```
var newgroup = nlapiCreateRecord('taxgroup', {nexuscountry: 'CA'});
newgroup.setFieldValue('itemid', 'CA-T5');
newgroup.setFieldValue('piggyback', 'T');
newgroup.setFieldValue('taxitem1', 21); //Canadian tax code
newgroup.setFieldValue('taxitem2', 24); //Canadian tax code
newgroup.setFieldValue('subsidiary', 2); //Canadian subsidiary
newgroup.setFieldValue('state', 'AB'); //Canadian subsidiary
newgroup.setFieldValue('description', 'New Canada Tax Group');

nlapiSubmitRecord(newgroup);
;
```

## Tax Period



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `taxperiod`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Preference UI Label	Preference Internal ID
First Fiscal Month	fiscalmonth
Fiscal Year Ene	fiscalyear
Period Format	periodstyle
Year in Period Name	periodname
One-Day Year-End Adjustment Period	lastday

## Tax Type

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `taxtype`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Term

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `term`. Search is not available on this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Unit of Measure

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `unitstype`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.





**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Vendor Category



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is vendorcategory. Search is not available on this record type.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Workplace



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is workplace.

This record is available only when the Payroll, Payroll Service, and Enhanced Premier Payroll features are enabled at Setup > Company > Enable Features on the Employees subtab.

In the user interface, you can access this record at Lists > Employees > Workplaces. For help working with this record in the user interface, see the help topic [Entering Workplace Records for Payroll](#).

## Supported Script Types

This record is scriptable in both client and server SuiteScript.

## Supported Functions

This record is fully scriptable, which means that the record can be created, updated, copied, deleted, and searched using SuiteScript. Refer to the following table for more details.

Function	Supported?
nlapiCopyRecord	yes
nlapiCreateRecord	yes
nlapiLoadRecord	yes
nlapiDeleteRecord	yes
nlapiSearchRecord	yes
nlapiSubmitRecord	yes

# Customization

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following customization records are scriptable in SuiteScript:

- Custom List
- Custom Segment Fields and Values
- Custom Transaction
- Scheduled Script Instance
- Script
- Script Deployment

## Custom List

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Each custom list will have a unique internal ID. For example, a custom list's internal ID might be customlist22, or customlist5, or customlist\_shirtColors, depending on whether you have accepted the default ID assigned to the custom list or you have created your own ID.

To see a list of IDs for all your custom lists, in the UI, go to Customization > Lists, Records, & Fields > Lists. If you have the Show Internal IDs preference enabled, all list internal IDs will appear in the ID column.

- Search Filters
- Search Columns

## Search Filters

Field Internal ID	Field UI Label	Field Type
internalid	Internal ID	select
internalidnumber	Internal ID (Number)	integer
isinactive	Inactive	checkbox
name	Name	text

## Search Columns

Field Internal ID	Field UI Label	Field Type
internalid	Internal ID	select
isinactive	Inactive	checkbox
name	Name	text



# Custom Segment Fields and Values

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Custom segments are custom classification fields similar to class, department, and location. When the Custom Segments feature is enabled, an administrator working in the UI can create custom segments, define possible values for each segment, and apply the segments to specific record types. A custom segment appears as a dropdown list or multiselect field on instances of record types where the segment was applied. Users can then classify records by selecting the appropriate value for each segment.

You cannot use SuiteScript to create a custom segment. However, you can use SuiteScript to create values for existing custom segments. You can also use SuiteScript to set values for segments that appear as fields on record instances. For details, see the following sections:

- [Using SuiteScript to Create Values for Existing Custom Segments](#)
- [Using SuiteScript to Set Values for Custom Segment Fields](#)

## Using SuiteScript to Create Values for Existing Custom Segments

Your NetSuite account contains a custom record type for every custom segment that has been defined. To add a value to the segment, you add an instance of this record type.

You can find the internal ID of the appropriate custom record type on the custom segment definition, under the Record ID label.

The screenshot shows a 'Custom Segment' creation form. At the top are buttons for 'Save', 'Cancel', 'Reset', and 'Manage Values'. Below is a 'Primary Information' section with a 'LABEL' field containing 'Profit Center' and an 'ID' field containing 'cseg\_profitcenter'. A red box highlights the 'RECORD ID' field, which contains 'customrecord\_cseg\_profitcenter'.

The following SuiteScript 1.0 snippet shows how to create a value for a segment called Profit Center.

```
...
// Create new segment value
var newSegmentValue = nlapiCreateRecord('customrecord_cseg_profitcenter');

// Set a name value on segment value record
newSegmentValue.setFieldValue('name', stlItemIdValue);

// If appropriate, you may want to set a parent value for the new value.
// For example, you might have a static value that you use for all
```

```
// segment values created through a script.  
newSegmentValue.setFieldText('parent', 'Created from automated script');  
  
...
```

## Using SuiteScript to Set Values for Custom Segment Fields

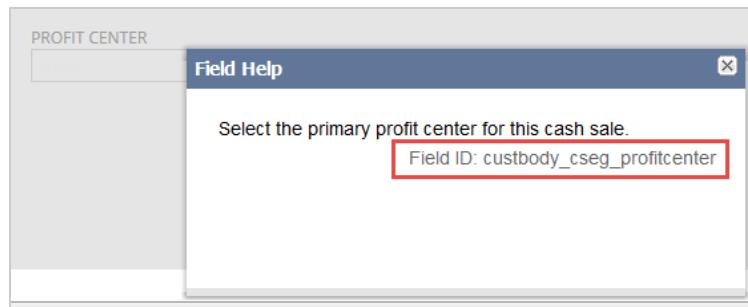
When a segment is applied to a record type, it appears as a field on instances of that type. You can interact with this field as you would a custom field. To do this, you need to be able to identify both the segment and the relevant values.

For details, see the following topics:

- Identifying a Segment Field
- Identifying a Segment Value
- Example

## Identifying a Segment Field

You identify a segment field in the same way you would a custom field: by using its field ID. To find the field ID, view an instance of the record type that you want to interact with. Click the label of the field to display a window that includes the field ID.



If you have permission to view custom segment definitions, you might notice that the segment definition lists a script ID and record ID. You should **not** refer to either of these IDs when setting a value for the segment on a record instance.

Additionally, be aware that a single custom segment can appear on instances of multiple record types. The field ID for the segment can vary among record types. You must use the field ID as it appears on an instance of the **same record type** where you want your script to set a segment value.

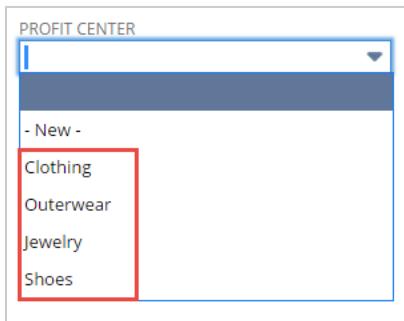
## Identifying a Segment Value

Typically, each custom segment includes a list of possible values. To set a value for a segment by using SuiteScript, at least one value must already have been defined for the segment.

You can reference a custom segment value by specifying either of the following:

- The value's text label
- The value's internal ID

The text label is shown in the custom segment definition and on instances of the record type where the segment has been applied. On the record instance, the segment's values are shown in either a multiselect box or a dropdown list.



To find a segment value's internal ID, open the segment definition at Customization > Lists, Records, & Fields > Custom Segments. To see a list of all possible values, refer to the Values sublist. This list includes a column labeled ID, which identifies each value's internal ID.

Values	Application & Sourcing	Validation & Defaulting	Permissions
DISPLAY ORDER X SUBLIST ALPHABETICAL			
ID	VALUE		
1	Clothing		
2	Outerwear		
3	Jewelry		
4	Shoes		

## Example

The following snippet shows how you could use SuiteScript 1.0 to set a value for a segment with the field ID of custbody\_cseg\_profitcenter:

```
nlapiSetFieldText('custbody_cseg_profitcenter', 'Outerwear');
```

## Custom Transaction



**Note:** The content in this help topic pertains to all versions of SuiteScript.

You use the custom transaction record to interact with instances of existing custom transaction types. For example, suppose you had a custom transaction type called Non-Operational Income Entry. In this case, you could use SuiteScript to create and modify non-operational income entries.

The internal ID for this record varies depending on the ID value of the transaction type. For example, if your transaction type had an ID value of \_noie, the SuiteScript internal ID would be customtransaction\_noie. You can view the transaction type's ID value by opening the type for editing. To open the type for editing, go to Customization > Lists, Records, & Fields > Transaction Types, and select the appropriate type.

## Supported Script Types

Custom transaction record instances are supported in both client and server SuiteScript.

All three user events are supported: beforeLoad, beforeSubmit, and afterSubmit.

## Supported Functions

Custom transaction records are fully scriptable. They can be created, updated, copied, searched, and deleted using SuiteScript. For more details, see the following table.

Function	Supported?	Notes
nlapiCopyRecord	Yes	
nlapiCreateRecord	Yes	
nlapiDeleteRecord	Yes	
nlapiLoadRecord	Yes	
nlapiSearchRecord	Yes	You use the standard transaction search to search custom transactions. For more information, see <a href="#">Transaction Search</a> .
nlapiSubmitRecord	Yes	
nlapiTransformRecord	No	
nlapiVoidTransaction	Yes	nlapiVoidTransaction is supported only for transaction types configured to support the void option. In other words, on the transaction type definition, the box labeled <b>Allow Void Transactions Using Reversal Journals</b> must be checked. Direct voids are never permitted with custom transactions. Only voids through reversal journals are permitted.

## Usage Notes

In the UI, custom transactions can have a field labeled Status. In SuiteScript, this field has an internal ID of transtatus. Values for transtatus are set by the system when the statuses are created. They cannot be modified. Possible values are A through Z. In the UI, you can view a transaction type's available statuses (and their transtatus values) by opening the transaction type and navigating to the Statuses subtab. The transtatus value for each status is listed in the Code column.

## Code Sample

The following sample shows how to create an instance of a custom transaction type.

```
var transaction = nlapiCreateRecord('customtransaction_mytype',true);
transaction.setFieldValue('subsidiary','1');

// Add debit line
```



```

transaction.selectNewLineItem('line');
transaction.setCurrentLineItemValue('line', 'account', '1');
transaction.setCurrentLineItemValue('line','debit',10);
transaction.setCurrentLineItemValue('line','credit',0);
transaction.setCurrentLineItemValue('line', 'memo', 'My first custom transaction line');
transaction.commitLineItem('line');

// Add credit line
transaction.selectNewLineItem('line');
transaction.setCurrentLineItemValue('line', 'account', '2');
transaction.setCurrentLineItemValue('line','debit',0);
transaction.setCurrentLineItemValue('line','credit',10);
transaction.setCurrentLineItemValue('line', 'memo', 'My second custom transaction line');
transaction.commitLineItem('line');

nlapiSubmitRecord(transaction);

```

## Scheduled Script Instance

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The scheduled script instance record type is viewed in the UI as the Scheduled Script Status page. The Scheduled Script Status page shows the scheduled script queue. Each script listed on this page is an instance of scheduled script instance. To access the Scheduled Script Status page in the UI, go to Customization > Scripting > Script Deployments > Status.

Scheduled Script Status							
<input type="button" value="Refresh"/> <input type="button" value="Print"/> <span style="float: right;">Total: 22</span>							
<input type="checkbox"/> <b>FILTERS</b>		DATE	FROM	TO	SCRIPT	DEPLOYMENT ID	
All					- All -	- All -	
customdeploy1	DATE CREATED	11.8.2014 12:01:07 am			STATUS	START	END
customdeploy1		Complete	12:01:44 am			12:01:51 am	100.0%
customdeploy1		9.9.2014 12:01:28 am				12:01:58 am	12:03:10 am
customdeploy1		Complete	12:02:26 am			12:02:35 am	100.0%
customdeploy1		13.8.2014 12:01:02 am				12:02:35 am	100.0%
customdeploy1		14.8.2014 12:01:18 am				12:02:46 am	100.0%
customdeploy1		Complete	12:01:51 am			12:02:06 am	100.0%
customdeploy1		15.8.2014 12:01:18 am				12:01:49 am	100.0%

To search scheduled script instance records, use 'scheduledscriptinstance' as the type argument.

For additional information, see:

- [Usage Notes](#)
- [Supported Functions](#)
- [Code Samples](#)

## Usage Notes

You can search scheduled script instance records with all server-side script types. Scheduled script instance records are not exposed to client-side scripts.



**Important:** You can only use the scheduled script instance record type to perform searches of the scheduled script queue.

The Cancel column is not exposed to SuiteScript.

To access scheduled script instance records with a script, the script owner must belong to a role assigned with SuiteScript permissions.

## Supported Functions

Use the following APIs to search scheduled script instance records:

- [nlapiCreateSearch\(type, filters, columns\)](#)
- [nlapiLoadSearch\(type, id\)](#)
- [nlapiSearchRecord\(type, id, filters, columns\)](#)

## Code Samples

The following code filters the search for all schedule script instance records that are in progress in queue 2. It uses search return columns to return all available information on the filtered results. Note that the first two search return columns are joins on the script and script deployment record types.

```
var colArr = new Array();
var filterArr = new Array();

colArr.push(new nlobjSearchColumn('name', 'script'));
colArr.push(new nlobjSearchColumn('internalid', 'scriptdeployment'));
colArr.push(new nlobjSearchColumn('datecreated'));
colArr.push(new nlobjSearchColumn('status'));
colArr.push(new nlobjSearchColumn('startdate'));
colArr.push(new nlobjSearchColumn('enddate'));
colArr.push(new nlobjSearchColumn('queue'));
colArr.push(new nlobjSearchColumn('percentcomplete'));
colArr.push(new nlobjSearchColumn('queueposition'));
colArr.push(new nlobjSearchColumn('percentcomplete'));
filterArr.push(new nlobjSearchFilter('queue', null, 'is', "2"));
filterArr.push(new nlobjSearchFilter('status', null, 'anyof', ['PROCESSING']));
var searchResults = nlapiSearchRecord('scheduledscriptinstance', null, filterArr, colArr);
```

## Script



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

To load and submit a script record, use one of the following case-insensitive script types for the type argument:

- 'bundleinstallationscript'

- 'clientscript'
- 'massupdatescript'
- 'portlet'
- 'restlet'
- 'scheduledscript'
- 'suitelet'
- 'usereventsheet'
- 'workflowactionsheet'

For the id argument, you must use the script record's internal ID (for example, 191); the script record's ID (for example, 'customscript\_csalert') is not supported. You can find a script record's internal ID at Customization > Scripting > Scripts. If you have the *Show Internal IDs* preference enabled at Home > Set Preferences, internal IDs are listed in the Internal ID column.

Scripts							<a href="#">View Deployments</a>
<a href="#">New Script</a>							
<a href="#">FILTERS</a>							Total: 46
EDIT	VIEW	NAME	FROM BUNDLE	ID	DEPLOYMENTS	SCRIPT	LIBRARY SCRIPT
View	SuiteSocial Tracked Fields Tab			43437	Deployments	suitesocial_user_tracked_fields.js	PSG-tools-common.js 250
View	SuiteSocial Tracked Field User Event			43437	Deployments	suitesocial_admin_assistant_auto_post_field_user_event.js	suitesocial_record_field_constants.js 292
View	SuiteSocial Tab			43437	Deployments	newsfeed_ui.js	PSG-tools-common.js 241

See the [SuiteScript Records Browser](#) for all other internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

Use `nlapiSearchRecord(type, id, filters, columns)`, `nlapiLoadSearch(type, id)` and `nlapiCreateSearch(type, filters, columns)` to perform searches on script records. Note that when you use one of these APIs to search for script records, you must pass in 'script' as the type parameter. This differs from how you load script records with `nlapiLoadRecord`; in those instances you pass in the individual script type.

For additional information about the script record, see:

- [Usage Notes](#)
- [Supported Functions](#)
- [Code Samples](#)

## Usage Notes

You can read and edit script records with all server-side script types. Script records are not exposed to client-side scripts.

**Important:** Script records cannot be created, copied or deleted programmatically.

The following objects are not scriptable, or are read only, on the script record:

- The Change ID button (in Edit view of UI) is not scriptable
- The Unhandled Errors tab is not scriptable
- The History tab is not scriptable
- The System Notes subtab is not scriptable

- The Execution Log tab is not scriptable
- The Libraries tab is not scriptable
- The Script Type field is read only; editing is not supported
- The Parameters tab is read only; editing is not supported

To access script records within a script, the script owner must belong to a role assigned with SuiteScript permissions.



**Important:** Scripts can only access script records when the records are part of the same bundle.

Scripts that are not part of a bundle cannot access script records that are part of a bundle.

## Supported Functions

Use the following APIs to load and submit script records.

- [nlapiLoadRecord\(type, id, initializeValues\)](#)
- [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#)

Use the following [nlobjRecord](#) methods to read and edit script records:

- [commitLineItem\(group, ignoreRecalc\)](#)
- [findLineItemValue\(group, fldnam, value\)](#)
- [getAllFields\(\)](#)
- [getAllLineItemFields\(group\)](#)
- [getCurrentLineItemDateTimeValue\(type, fieldId, timeZone\)](#)
- [getCurrentLineItemValues\(type, fldnam\)](#)
- [getField\(fldnam\)](#)
- [getFieldText\(name\)](#)
- [getFieldValue\(name\)](#)
- [getId\(\)](#)
- [getLineItemCount\(group\)](#)
- [getLineItemDateTimeValue\(type, fieldId, lineNum, timeZone\)](#)
- [getLineItemField\(group, fldnam, linenum\)](#)
- [getLineItemText\(group, fldnam, linenum\)](#)
- [getLineItemValue\(group, name, linenum\)](#)
- [getLineItemValues\(type, fldnam, linenum\)](#)
- [getRecordType\(\)](#)
- [insertLineItem\(group, linenum, ignoreRecalc\)](#)
- [removeLineItem\(group, linenum, ignoreRecalc\)](#)
- [selectLineItem\(group, linenum\)](#)
- [selectNewLineItem\(group\)](#)
- [setCurrentLineItemDateTimeValue\(type, fieldId, dateTime, timeZone\)](#)

- `setCurrentLineItemValue(group, name, value)`
- `setFieldText(name, text)`
- `setFieldValue(name, value)`
- `setLineItemDateTimeValue(type, fieldId, lineNum, date, time, timeZone)`
- `setLineItemValue(group, name, lineNum, value)`

Use the following Search APIs to perform searches on script records:

- `nlapiCreateSearch(type, filters, columns)`
- `nlapiLoadSearch(type, id)`
- `nlapiSearchRecord(type, id, filters, columns)`

## Code Samples

```
//read a suitelet script record
var rec = nlapiLoadRecord('suitelet',605);
var a = rec.getFieldValue('scriptfile');
var count = rec.getLineItemCount('parameters');
var pLabel = rec.getLineItemValue('parameters', 'label', 1);
var pld = rec.getLineItemValue('parameters', 'internalid', 1);
var pType = rec.getLineItemValue('parameters', 'fieldtype', 1);
var pRecordType = rec.getLineItemValue('parameters', 'selectrecordtype', 1);
```

```
//edit a user event script record
var rec = nlapiLoadRecord('userevents', 302);
rec.setFieldValue('name', 'userevent_001');
rec.setFieldValue('scriptfile', '227');
rec.setFieldValue('notifyadmins', 'T');
rec.setFieldValue('aftersubmitfunction', 'afterSubmitFunction');
nlapiSubmitRecord(rec);
```

```
//search a script record
var filters = [new nlobjSearchFilter('defaultfunction', null, 'is', 'myfunctionname')];
var columns = [new nlobjSearchColumn('name')];
var results = nlapiSearchRecord('script', null, filters, columns);
```

## Script Deployment

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

To load, submit, copy, create and delete a script deployment record, use 'scriptdeployment' for the type argument. Note that 'scriptdeployment' is case-insensitive. For the id argument, you must use the script deployment record's internal ID (for example, 88); the script deployment record's ID (for example, 'customdeploy\_newsfeed\_email') is not supported. You can find a script deployment record's internal ID at Customization > Scripting > Script Deployments. If you have the Show Internal IDs preference enabled at Home > Set Preferences, internal IDs are listed in the Internal ID column.

Script Deployments							
FILTERS		Scripts					
TYPE	STATUS	RECORD TYPE	SCRIPT	STATUS	TITLE	SCHEDULE	LAST MODIFIED
Scheduled	- All -	- All -	- All -	- All -	- All -	- All -	- All -
	<input type="checkbox"/> Show Undeployed						Total: 45
INTERNAL ID	EDIT   HOW	ID	SCRIPT	STATUS	TITLE	SCHEDULE	LAST MODIFIED
382	Edit   View	customdeploy_scm_ss_bgprocess	SCM Background Process SS	Not Scheduled	SCM Background Process SS	one time event on 4.9.2013	25.9.2013 11:54 pm
475	Edit   View	customdeploy_advpromo_datamigrate_ss	Advanced Promotion Data Migrator	Not Scheduled	Advanced Promotion Data Migrator	one time event on 11.3.2014	20.3.2014 7:48 pm
212	Edit   View	customdeploy2	LNS_Status	Not Scheduled	LNS_Status 2	one time event on 20.3.2009	3.4.2013 10:11 am

See the [SuiteScript Records Browser](#) for all other internal IDs associated with this record. For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

For additional information about the script deployment record, see:

- [Usage Notes](#)
- [Supported Functions](#)
- [Code Samples](#)

## Usage Notes

You can access script deployment records with all server-side script types. Script deployment records are not exposed to client-side scripts.

The following objects are not scriptable, or are read only, on the script deployment record:

- Change ID button (in Edit view of UI) is not scriptable
- The History subtab is not scriptable
- The System Notes subtab is not scriptable
- The Execution Log subtab is not scriptable
- The Scheduling subtab is not scriptable
- The Parameters subtab is read only; editing is not supported

To access script deployment records within a script, the script owner must belong to a role assigned with SuiteScript permissions.



**Important:** Scripts can only access script deployment records when the records are part of the same bundle.

Scripts that are not part of a bundle cannot access script deployment records that are part of a bundle.

## Supported Functions

Use the following APIs to copy, create, delete, load and submit script deployment records.

- [nlapiCopyRecord\(type, id, initializeValues\)](#)
- [nlapiCreateRecord\(type, initializeValues\)](#)
- [nlapiDeleteRecord\(type, id, initializeValues\)](#)
- [nlapiLoadRecord\(type, id, initializeValues\)](#)

- nlapiSubmitRecord(record, doSourcing, ignoreMandatoryFields)

Use the following `nlobjRecord` methods to read and edit script deployment records:

- commitLineItem(group, ignoreRecalc)
- findLineItemValue(group, fldnam, value)
- getAllFields()
- getAllLineItemFields(group)
- getCurrentLineItemDateTimeValue(type, fieldId, timeZone)
- getCurrentLineItemValues(type, fldnam)
- getField(fldnam)
- getFieldText(name)
- getFieldValue(name)
- getId()
- getLineItemCount(group)
- getLineItemDateTimeValue(type, fieldId, lineNum, timeZone)
- getLineItemField(group, fldnam, linenum)
- getLineItemText(group, fldnam, linenum)
- getLineItemValue(group, name, linenum)
- getLineItemValues(type, fldnam, linenum)
- getRecordType()
- insertLineItem(group, linenum, ignoreRecalc)
- removeLineItem(group, linenum, ignoreRecalc)
- selectLineItem(group, linenum)
- selectNewLineItem(group)
- setCurrentLineItemDateTimeValue(type, fieldId, dateTime, timeZone)
- setCurrentLineItemValue(group, name, value)
- setFieldText(name, text)
- setFieldValue(name, value)
- setLineItemDateTimeValue(type, fieldId, lineNum, dateTime, timeZone)
- setLineItemValue(group, name, linenum, value)

Use the following Search APIs to perform searches on script deployment records:

- nlapiCreateSearch(type, filters, columns)
- nlapiLoadSearch(type, id)
- nlapiSearchRecord(type, id, filters, columns)

## Code Samples

```
//create a script deployment record
var rec = nlapiCreateRecord('scriptdeployment', {'script':'2'});
```



```
//edit a script deployment record
var rec = nlapiLoadRecord('scriptdeployment',65);
rec.setFieldValue('allroles', 'T');
nlapiSubmitRecord(rec);
```

```
//read a script deployment record
var rec = nlapiLoadRecord('scriptdeployment', 203);
var a = rec.getFieldValue('status')
```

```
//copy and edit a script deployment record
var rec = nlapiCopyRecord('scriptdeployment', 41);
rec.setFieldValue('isdeployed', 'F');
rec.setFieldValue('title', 'Feature 4141 - Suitelet Scratch 4');
nlapiSubmitRecord(rec);
```

```
//search a script deployment record
var filters = [new nlobjSearchFilter('internalidnumber', null, 'anyof', '43')];
var columns = [new nlobjSearchColumn('title')];
var results = nlapiSearchRecord('scriptdeployment', null, filters, columns);
```



# Marketing

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following marketing records are scriptable in SuiteScript:

- Campaign
- Campaign Template
- Coupon Code
- Email Template
- Promotion

## Campaign

**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is `campaign`.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Campaign Template

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The campaign template record is used for both CRMSDK templates and scriptable templates. In the UI, you find this record by choosing Documents > Templates > Marketing Templates

Scriptable templates are not supported with SuiteScript.

The internal ID for this record is `campaigntemplate`.

## Supported Script Types

This record is only supported in server-side scripts.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

You can use `nlapiMergeRecord(id, baseType, baselId, altType, altId, fields)` with CRMSDK templates to perform a mail merge. You can also use SuiteScript to read and delete CRMSDK templates. You cannot create, edit, copy, search, or transform CRMSDK templates with SuiteScript.



## Field Definitions

This section describes some of the key fields on the campaign template record.

### Content and Mediaitem

When accessing this record, only one of the following fields contains a value:

- **content** — This field maps to the text editor that appears on the Template tab in the UI. This field contains plain text or HTML.
- **mediaitem** — This field maps to the File text box that appears on the Template tab in the UI. This field identifies the file that is used as the basis for the template.

### For More Information

See the [SuiteScript Records Browser](#) for the internal IDs of all fields associated with this record, their corresponding labels in the UI, and more details. Fields not listed in the Records Browser are not supported and should not be used.

For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).

## Code Sample

The following sample shows how to access a campaign template email.

### Accessing Template Content

The following sample shows how to access template content:

```
var template = nlapiLoadRecord('campaigntemplate', 154);

if (template.getFieldValue('mediaitem') != null) {
    var media = template.getFieldValue('mediaitem');
    // do something with the media...
} else {
    var content = template.getFieldValue('content');
    // do something with the content...
}
```

## Coupon Code



**Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is couponcode.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Email Template



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The email template record is used for both CRMSDK templates and scriptable templates. In the UI, you find this record by choosing Documents > Templates > Email Templates. This record is available only if the Customer Relationship Management feature has been enabled at Setup > Company > Enable Features, on the CRM tab.

Scriptable templates are not supported with SuiteScript.

The internal ID for this record is `emailtemplate`.

## Supported Script Types

This record is only supported in server-side scripts.

All three user events are supported: `beforeLoad`, `beforeSubmit`, and `afterSubmit`.

## Supported Functions

You can use `nlapiMergeRecord(id, baseType, baseld, altType, altld, fields)` with CRMSDK templates to perform a mail merge. You can also use SuiteScript to read and delete CRMSDK templates. You cannot create, edit, copy, search, or transform CRMSDK templates with SuiteScript.

## Field Definitions

This section describes some of the key fields on the email template record.

## Content and Mediaitem

When accessing this record, only one of the following fields contains a value:

- `content` — This field maps to the text editor that appears on the Template tab in the UI. This field contains plain text or HTML.
- `mediaitem` — This field maps to the File text box that appears on the Template tab in the UI. This field identifies the file that is used as the basis for the template.

## For More Information

See the [SuiteScript Records Browser](#) for the internal IDs of all fields associated with this record, their corresponding labels in the UI, and more details. Fields not listed in the Records Browser are not supported and should not be used.



For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#).

## Code Sample

The following example shows how you can script with email template records.

## Accessing Template Content

The following sample shows how to access template content:

```
var template = nlapiLoadRecord('emailtemplate', 124);

if (template.getFieldValue('mediaitem') != null) {
    var media = template.getFieldValue('mediaitem');
    // do something with the media...
} else {
    var content = template.getFieldValue('content');
    // do something with the content...
}
```

## Promotion

 **Note:** The content in this help topic pertains to all versions of SuiteScript.

The internal ID for this record is promotioncode.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific fields on this record.

Field Internal ID	Field UI Label	Note
<b>Body Fields</b>		
discounttype	radio buttons	Valid values in scripts are: <ul style="list-style-type: none"> <li>■ percent</li> <li>■ flat</li> </ul>

# Website

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following website records are scriptable in SuiteScript:

- Web Site Setup
- Commerce Category

## Web Site Setup

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `website`. In the UI, you can find this record by going to **Setup > Web Site > Set Up Web Site**.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The Web Site Setup (`website`) record supports server side SuiteScripts such as user event scripts. Note, however, this record does not support beforeLoad user event scripts.

Developers can use SuiteScript with the Web Site Setup record whether they are working in accounts using Site Builder or SuiteCommerce Advanced.

## Creating a Web Site Setup Record in SuiteScript

To create a Web Site Setup record in a **Site Builder** site, developers can write code using one of two approaches:

The first approach is:

```
var recWebSite = nlapiCreateRecord('website');
```

The second approach is:

```
var initvalues = new Array();
initvalues.sitetype = 'STANDARD';
var recWebSite = nlapiCreateRecord('website', initvalues);
```

In the first approach, the `initvalues` are defaulted to STANDARD. However, it is recommended that developers use the second approach because there are two web site types. To clearly distinguish between the two types in your code, the second approach is cleaner.

To create a Web Site Setup record in a **SuiteCommerce Advanced** site, the code **must** look like the following:

```
var initvalues = new Array();
initvalues.sitetype = 'ADVANCED';
var recWebSite = nlapiCreateRecord('website', initvalues);
```

## Setting Values for Web Site Setup Dropdown Fields

The following table lists the dropdown fields on the Web Site Setup record. When writing SuiteScript, if you are setting the value of a dropdown field, use the IDs listed in the column called Internal IDs for Dropdown Values.

Tab	Dropdown Field	UI Labels for Dropdown Values	Internal IDs for Dropdown Values
<b>Setup tab</b>			
	Web Site Scope (websitetoscope)	<ul style="list-style-type: none"> <li>■ Full Web Store</li> <li>■ Information And Catalog, With Pricing</li> <li>■ Information And Catalog</li> <li>■ Information Only</li> </ul>	<ul style="list-style-type: none"> <li>■ FULL_WEB_STORE</li> <li>■ INFO_CATALOG_PRICING</li> <li>■ INFO_CATALOG</li> <li>■ INFO_ONLY</li> </ul>
	Default Customer Category (defaultcustomercategory)	<ul style="list-style-type: none"> <li>■ Corporate</li> <li>■ Individual</li> <li>■ Employee</li> </ul>	<ul style="list-style-type: none"> <li>■ CORPORATE</li> <li>■ INDIVIDUAL</li> <li>■ EMPLOYEE</li> </ul>
<b>Appearance tab</b>			
	Web Site Logo Alignment (websitelogoalign)	<ul style="list-style-type: none"> <li>■ Align Left</li> <li>■ Align Right</li> <li>■ Align Center</li> </ul>	<ul style="list-style-type: none"> <li>■ LEFT</li> <li>■ RIGHT</li> <li>■ CENTER</li> </ul>
	Page Alignment (pagealign)	<ul style="list-style-type: none"> <li>■ Align Left</li> <li>■ Align Right</li> <li>■ Align Center</li> </ul>	<ul style="list-style-type: none"> <li>■ LEFT</li> <li>■ RIGHT</li> <li>■ CENTER</li> </ul>
	Display Order of Cart Items (cartdisplayorder)	<ul style="list-style-type: none"> <li>■ Most Recently Added First</li> <li>■ Most Recently Added Last</li> </ul>	<ul style="list-style-type: none"> <li>■ RECENT_FIRST</li> <li>■ RECENT_LAST</li> </ul>
<b>Upsell tab</b>			
	Items to Upsell (upsellitems)	<ul style="list-style-type: none"> <li>■ Show Related Items First and Upsell Items Next</li> <li>■ Show Upsell Items First and Related Items Next</li> <li>■ Show Only Related Items</li> </ul>	<ul style="list-style-type: none"> <li>■ RELATED_FIRST_UPSELL_NEXT</li> <li>■ UPSELL_FIRST_RELATED_NEXT</li> <li>■ ONLY_RELATED_ITEMS</li> <li>■ ONLY_UPSELL_ITEMS</li> </ul>



Tab	Dropdown Field	UI Labels for Dropdown Values	Internal IDs for Dropdown Values
		<ul style="list-style-type: none"> <li>■ Show Only Upsell Items</li> </ul>	
	Items to Upsell in Cart (cartupsellitems)	<ul style="list-style-type: none"> <li>■ Show Related Items First and Upsell Items Next</li> <li>■ Show Upsell Items First and Related Items Next</li> <li>■ Show Only Related Items</li> <li>■ Show Only Upsell Items</li> </ul>	<ul style="list-style-type: none"> <li>■ RELATED_FIRST_UPSELL_NEXT</li> <li>■ UPSELL_FIRST_RELATED_NEXT</li> <li>■ ONLY_RELATED_ITEMS</li> <li>■ ONLY_UPSELL_ITEMS</li> </ul>
<b>Legacy tab</b>			
	Site Tab Alignment (sitetabalignment)	<ul style="list-style-type: none"> <li>■ Align Left</li> <li>■ Align Right</li> <li>■ Align Center</li> </ul>	<ul style="list-style-type: none"> <li>■ LEFT</li> <li>■ RIGHT</li> <li>■ CENTER</li> </ul>
<b>Shopping tab</b>			
	Sales Order Type (salesordertype)	<ul style="list-style-type: none"> <li>■ Per Customer Basis</li> </ul>	<ul style="list-style-type: none"> <li>■ PER_CUSTOMER</li> </ul>

## Commerce Category

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this record is `commercecategory`. In the UI, you can go to Lists > Web Site > Commerce Categories to access this record.

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

You must enable the Commerce Category feature on the Web Presence subtab at Setup > Company > Enable Features to be able to script with this record.

## Supported Script Types

The commerce category record supports server side SuiteScripts such as a script that searches for a category record by URL and site id.



The following scripting functions are not supported for this record:

- Attach
- Transform
- Copy

## Script Sample

The following sample code snippet searches for a commerce category record with a fullurl of "/cat-0" and siteid of 2. It returns fullurl, addtohead, and subcataddtoheadoverride as search result columns.

```
var fullurl = '/cat-0';
var siteid = 2;

var searchFilters = [
    new nlobjSearchFilter('fullurl', null, 'is', fullurl)
    , new nlobjSearchFilter('site', null, 'is', siteid)
];
var searchColumns = [
    new nlobjSearchColumn('fullurl')
    , new nlobjSearchColumn('addtohead')
    , new nlobjSearchColumn('subcataddtoheadoverride')
];
var search = nlapiSearchRecord('commercecATEGORY', null, searchFilters, searchColumns);
```

# Scriptable Sublists



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

## Scriptable Sublists Overview

The following table lists all NetSuite sublists that support SuiteScript. The internal ID for each sublist is also listed. When using [Sublist APIs](#), you will need to pass the internal ID of the sublist, as well the internal IDs of specific sublist fields you may be referencing in your scripts.

The internal IDs of all supported sublist fields are provided in the [SuiteScript Records Browser](#), which is organized by record type. If you know which record a sublist appears on, go to that record in the SuiteScript Records Browser. If you do not know which record the sublist appears on, click one of the sublist links in the table below. The documentation for each sublist states which record(s) the sublist appears on.

Important Things to Note:

1. Not every sublist field that appears in the [SuiteScript Records Browser](#) is settable through scripting. Some of the fields that appear are read-only. You will need to reference the NetSuite UI to know whether a field is settable. The general rule is that if you can set a field in the UI, you can set it in SuiteScript. If you cannot set a field in the UI, you cannot set it using SuiteScript. You can, however, still get the field's value using SuiteScript.
2. The following table lists only the sublists that officially support SuiteScript. Scripts that reference sublists which do not appear in this table may not run as intended. Even if scripts do run as intended, there is no guarantee they will continue to run in future versions of NetSuite if backend infrastructure changes are made to support a new feature. Therefore, it is imperative that your scripts reference only the sublists and sublist fields listed in this section and in the [SuiteScript Records Browser](#).

Sublist	Sublist Internal ID	Sublist Type
Access Sublist (contact roles)	contactroles	list
Accounts Sublist	directdeposit	inlineeditor
Accrued Time Sublist	accruedtime	inlineeditor
Adjustments Sublist	inventory	inlineeditor
Apply Sublist	apply	list
Assignees Sublist	assignee	inlineeditor
Attendees Sublist	attendee	inlineeditor
Billable Expenses Sublist	expcost	list
Billable Items Sublist	itemcost	list
Billable Time Sublist	time	list
Bin Numbers Sublist	binnumber	inlineeditor
Company Contributions Sublist	companycontribution	inlineeditor

Sublist	Sublist Internal ID	Sublist Type
Company Contributions Sublist	companytax	inlineeditor
Competitors Sublist	competitors	inlineeditor
Credits Sublist	credit	list
Currencies Sublist	currency	inlineeditor
Custom Child Record Sublists	recmachcustrecord	inlineeditor
Deductions Sublist	deduction	inlineeditor
Demand Plan Detail Sublist	demandplandetail	list
Deposits Sublist	deposit	list
Direct Mail Sublist	campaigndirectmail	inlineeditor
Download Sublist	download	inlineeditor
Earnings Sublist	earning	inlineeditor
E-mail Sublist	campaignemail	inlineeditor
Employee Taxes Sublist	employetax	inlineeditor
Escalate To Sublist	escalateto	inlineeditor
Expenses Sublist	expense	inlineeditor
Group Pricing Sublist	grouppricing	inlineeditor
Item Fulfillment/Receipt Sublist	item	list
Items Sublist	item	inlineeditor
Item Pricing Sublist	itempricing	inlineeditor
Lead Nurturing Sublist	campaigndrip	inlineeditor
Line Sublist	line	inlineeditor
Members Sublist	member	inlineeditor
Orders Sublist	order	inlineeditor
Other Events Sublist	campaignevent	inlineeditor
Partners Sublist	partners	inlineeditor
Pricing Sublist	price	list
Predecessors Sublist	predecessor	inlineeditor
Related Solutions Sublist	solutions	inlineeditor
Resources Sublist	resource	inlineeditor
Sales Team Sublist	salesteam	inlineeditor
Shipping Sublist	shipgroup	inlineeditor
Site Category	sitecategory	inlineeditor
Time Tracking Sublist	timeitem	inlineeditor
Topics Sublist	topics	inlineeditor
Units	uom	inlineeditor



Sublist	Sublist Internal ID	Sublist Type
Vendors	itemvendor	inlineeditor

## Access Sublist (contact roles)

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **contactroles**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

If you have upgraded the forms in your account, **contactroles** maps to the Access sublist, which appears on the System Information subtab. The Access sublist appears on the following record types: Partner, Prospect, Lead, Customer.

The following fields appear on the **contactroles** sublist. (All fields and internal IDs will soon be added to the SuiteScript Records Browser.)

Field Internal ID	Field UI Label	Type	Required
contact	Contact	select	false
email	Email	email	false
giveaccess	Access	checkbox	false
passwordconfirm	Confirm Password	password	false
role	Role	select	false
sendemail	Notify	checkbox	false

See the [SuiteScript Records Browser](#) for all internal IDs associated with this record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Accounts Sublist

The internal ID for this sublist is **directdeposit**. It is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Accounts sublist appears on the ACH/Direct Deposit subtab on the employee record. To see the internal IDs associated with the Accounts sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Accrued Time Sublist

The internal ID for this sublist is **accruedtime**. It is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Accrued Time sublist appears on the Payroll subtab on the employee record. To see the internal IDs associated with the Accrued Time sublist, refer to the [SuiteScript Records Browser](#).

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Adjustments Sublist

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **inventory**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Adjustments sublist appears on the inventory adjustment record. To see the internal IDs associated with the Adjustments sublist, refer to the [SuiteScript Records Browser](#).

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Apply Sublist

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **apply**. This is a **list** sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Apply sublist appears on the following records: Credit Memo, Customer Payment, Customer Refund, Deposit Application, Vendor Credit, Vendor Payment. To see the internal IDs associated with the Apply sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

 **Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

 **Important:** When the Apply sublist appears on the Customer Payment record, the Apply tab is still visible, however, all Apply sublist data appears on a tab called Invoice. Also note that the UI label for the **Amt. Due** (due) field can also appear as **Amount Remaining**.

## Assignees Sublist

 **Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **assignee**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Assignee sublist appears on the project task record. To see the internal IDs associated with the Assignees sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Attendees Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **attendee**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Attendees sublist appears on the event record. To see the internal IDs associated with the Attendees sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Billable Expenses Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **expcost**. This sublist is a **list** sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Billable Expenses sublist appears on the Cash Sale and Invoice records. To see the internal IDs associated with the Billable Expenses sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Billable Items Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **itemcost**. This is a **list** sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Billable Items sublist appears on Cash Sale and Invoice records. To see the internal IDs associated with the Billable Items sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Billable Time Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **time**. This is a [list](#) sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Billable Time sublist appears on the Cash Sale and Invoice records. To see the internal IDs associated with the Billable Time sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Bin Numbers Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **binnumber**. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Bin Numbers sublist appears on the Inventory Item record. To see the internal IDs associated with the Bin Numbers sublist, refer to the [SuiteScript Records Browser](#).

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Company Contributions Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **companycontribution**. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Company Contributions sublist appears on the [Paycheck Journal](#) record and the Employee record. To see the internal IDs associated with the Company Contributions sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Company Taxes Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **companytax**. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Company Taxes sublist appears on the [Paycheck Journal](#) record. To see the internal IDs associated with the Company Taxes sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Competitors Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **competitors**. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Competitors sublist appears on the Opportunity record. To see the internal IDs associated with the Competitors sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Credits Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **credit**. This is a [list](#) sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Credits sublist appears on the customer payment and vendor payment records. (Note that although the Credits sublist is supported on the Vendor Payments record type, this sublist is not

currently showing on this record in the vendor payment reference page in the Records Browser. To get the internal IDs for the Credits sublist, refer to the Records Browser's [customer payments](#) reference page.)

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Currencies Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **currency**. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Currencies sublist appears on the following record types: Customer, Lead, Prospect. To see the internal IDs associated with the Currencies sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Custom Child Record Sublists

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

When working with custom child record sublists you can use all the standard [Sublist APIs](#) provided in SuiteScript. If you are not familiar with custom child record sublists, it is recommended that you read these topics in order:

- [What Are Custom Child Record Sublists?](#)
- [Creating Custom Child Record Sublists](#)
- [Understanding Custom Child Record Sublist IDs](#)
- [Scripting with Custom Child Record Sublists](#)

## What Are Custom Child Record Sublists?

Custom child record sublists are [Inline Editor Sublists](#) that contain a list of custom records.

The following figure shows a custom child record sublist of Fixed Assets records. These records appear as line items on a custom Fixed Assets subtab. The parent record that contains the sublist of custom Fixed Assets records is the Customer record.

In the UI, click **New Fixed Assets** to create a new Fixed Assets child record. The new record is added to the Fixed Assets sublist and is scriptable. In SuiteScript, add a new Fixed Assets child record by adding a new line to the sublist.

FIXED ASSETS NAME	COST *	USEFUL LIFE IN YEARS *
Fixed asset 1	200.00	10
Fixed asset 2	400.00	2
Fixed asset 3	1000.00	3
	1	

**Add** **Cancel** **Insert**

When the **New Fixed Assets** button is clicked, a new Fixed Assets record opens (see figure below). Note that the Fixed Assets record contains a **New Customer** field. This field is a List/Record field that references the parent record – in the case, the Abe Simpson customer record.

**Note:** The parent-child relationship between the Fixed Assets record type and the Customer record type was defined on the Custom Record Type definition page for the Fixed Assets record. (For general information on creating parent-child relationships between records, see the help topic [Understanding Parent-Child Record Relationships](#) in the *SuiteBuilder Guide*.)

The Fixed Assets fields that appears in the sublist are the mandatory fields (those body fields that appear with the yellow asterisk on the Fixed Asset record) and those fields that have been set to **Show in List** in the Custom Field definition page for the Fixed Assets record type. You can use [Sublist APIs](#) to set or get values for all Fixed Assets fields in a Fixed Assets sublist.

**Fixed Assets**

**Save** **Cancel** **Reset** **List** **Search** **Customize** **More**

<b>CUSTOM FORM *</b> Custom Fixed Assets Form 2	<b>SALVAGE VALUE</b>	<b>DESCRIPTION</b>
<b>NAME *</b>	<b>DEPRECABLE COST</b>	<b>NEW CUSTOMER</b>
<input type="checkbox"/> INACTIVE	<b>USEFUL LIFE IN YEARS *</b> 1	Abe Simpson
<b>COST *</b>		

**Notes** **Files** **Remove All**

TITLE	MEMO *	DATE	TIME	TYPE	DIRECTION
		9/9/2015	1:18 pm		

**Add** **Cancel** **Insert** **Remove**

**Save** **Cancel** **Reset**

The New Customer field is a List/Record field that references the parent record of the Fixed Assets (child) record.

As long as you know the internal ID of a Fixed Assets field, you can update that field through sublist scripting. (See [Understanding Custom Child Record Sublist IDs](#) to learn how to get field IDs for all fields on a custom child record sublist.)

The screenshot shows the 'Customer' record for 'Abe Simpson'. At the top, there are buttons for Save, Cancel, Reset, Merge, and Actions. Below that is the 'Primary Information' section with fields for CUSTOM FORM (set to 'Custom Customer Form 4'), PARENT COMPANY (dropdown), STATUS (set to 'CUSTOMER-Closed Won'), SALES REP (set to 'Jon Baker'), and CATEGORY (set to 'From advertisement'). The 'Email | Phone | Address' section contains fields for EMAIL ('asimpson@netsuite.com') and PHONE ('504-231-1111'). A red box highlights the 'Fixed Assets' tab in the navigation bar, which is currently selected. The 'Fixed Assets' sublist shows a table with three rows of data:

EDIT	NAME	COST	USEFUL LIFE IN YEARS	REMOVE
Edit	2003 Ford Expedition	32,885.00	5	<a href="#">Remove</a>
Edit	Dell Dimension XPS	1,649.00	3	<a href="#">Remove</a>
Edit	Printer	1,000.00	3	<a href="#">Remove</a>

Fields set to **Show in List** on the Fixed Assets record type appear in the sublist. However, you can update any Fixed Asset field via scripting.

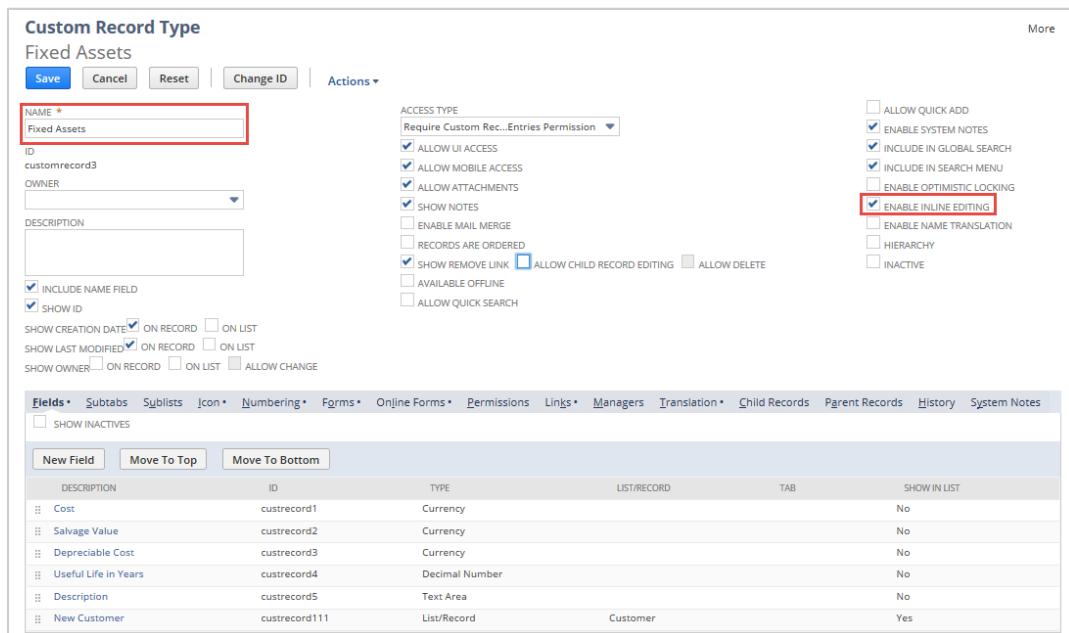
## Creating Custom Child Record Sublists

The following steps are high-level. They are provided for users who are already familiar with NetSuite customization, but who need a general frame of reference for building a custom child record sublist. Detailed steps for creating custom records types, custom fields, and custom subtabs are provided in the *SuiteBuilder Guide*.

### To create a custom child record sublist:

1. Define a custom record type (such as the Fixed Assets record mentioned earlier).

2.  **Important:** On the Custom Record Type page, select Allow Inline Editing (see figure). If this preference is not enabled, your records will not be scriptable when they appear as sublist line items on the parent record.



DESCRIPTION	ID	TYPE	LIST/RECORD	TAB	SHOW IN LIST
Cost	custrecord1	Currency			No
Salvage Value	custrecord2	Currency			No
Depreciable Cost	custrecord3	Currency			No
Useful Life in Years	custrecord4	Decimal Number			No
Description	custrecord5	Text Area			No
New Customer	custrecord111	List/Record	Customer		Yes

3. Establish the parent-child relationship between your new custom record type (Fixed Assets) and another record type. Parent-child relationships are established through custom fields.
1. Add a custom field to your new custom record type.
  2. On the field definition page for the new field, set the field **Type** to **List/Record** (see figure below).
  3. Specify the record type that will be the parent of your custom record type. In the following figure, the Customer record type will be the parent.
  4. Select the **Record is Parent** check box. Doing so attaches your custom record type (Fixed Assets) to a parent record type (Customer).

In this case, it is the **New Customer** field that ties the Fixed Assets record type to the Customer record type.

The screenshot shows the 'Fixed Assets Field' configuration page. At the top, there are buttons for Save, Cancel, Reset, Change ID, Apply to Forms, and Actions. On the left, fields include Label (New Customer), ID (custrecord111), Internal ID (1478), and Owner (45 Wolfe, K). On the right, there's a 'DESCRIPTION' field and several checkboxes: SHOW IN LIST (checked), GLOBAL SEARCH, RECORD IS PARENT (checked), INACTIVE, and APPLY ROLE RESTRICTIONS. Below these are tabs for Display, Validation & Defaulting, Sourcing & Filtering, Access, Translation, and History. Under 'Display', settings include Insert Before (- Unchanged -), Subtab (empty), and Display Type (Normal). Under 'Access', there's a 'HELP' field and a 'PARENT SUBTAB' dropdown set to 'Fixed Assets', which is also highlighted with a red box. A note at the bottom says 'ALLOW QUICK ADD'.

You can have your custom child record sublist appear on a standard or custom subtab of the parent record.

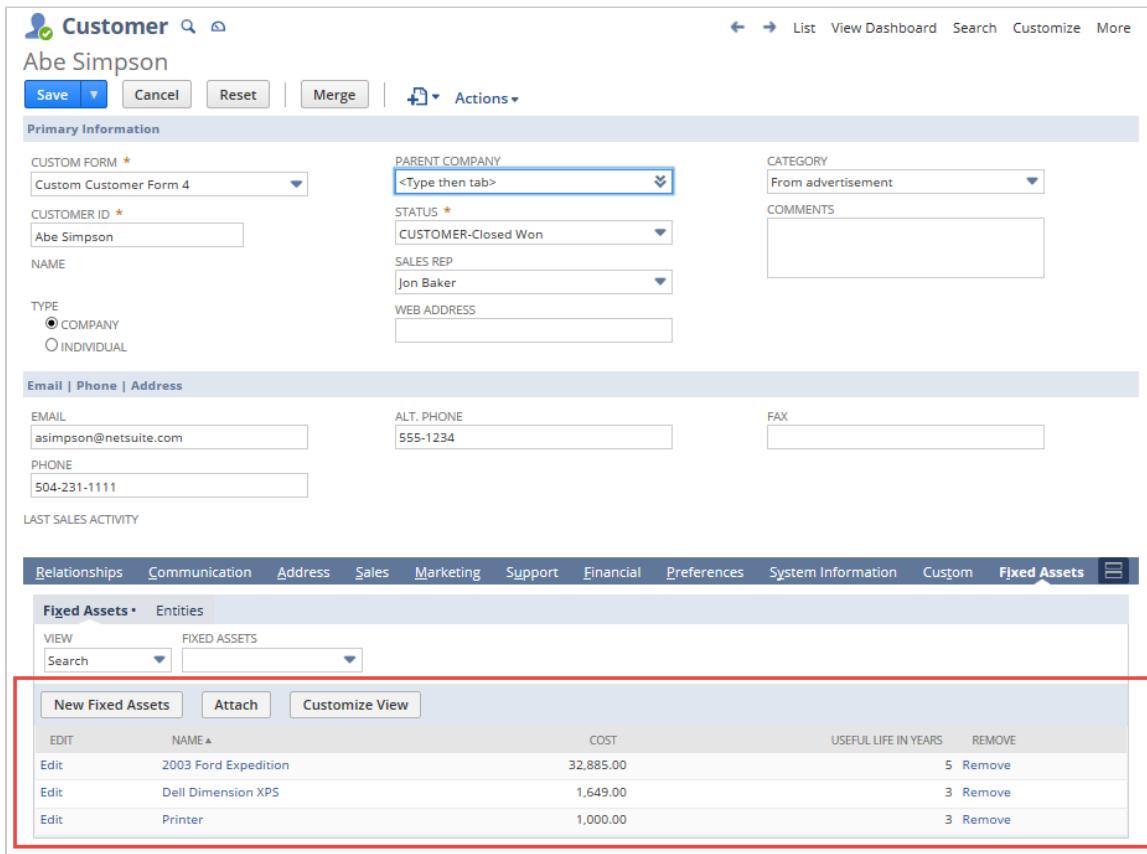
- If you want the custom child record sublist to appear on its own subtab on the parent record, create a subtab with a name that reflects the sublist type. The figure above shows that a sublist of child Fixed Asset records will appear on a custom **Fixed Assets** subtab. This subtab will appear on all Customer (parent) records.



**Note:** See the *SuiteBuilder Guide* for steps on creating custom subtabs and adding subtabs to specific record types.

- After defining the Customer–Fixed Assets (parent-child) relationship (via the **New Customer** field), go to a Customer record in NetSuite and notice the Fixed Assets sublist (see figure).

**Note:** If you have not specified a parent subtab for this sublist, the custom record child sublist will appear on a system-generated subtab called **Custom**.



The screenshot shows the 'Customer' record for 'Abe Simpson'. The 'Fixed Assets' subtab is selected. A red box highlights the 'New Fixed Assets' button and the sublist table below it. The table lists three fixed assets with columns for Edit, NAME, COST, USEFUL LIFE IN YEARS, and REMOVE.

EDIT	NAME	COST	USEFUL LIFE IN YEARS	REMOVE
Edit	2003 Ford Expedition	32,885.00	5	Remove
Edit	Dell Dimension XPS	1,649.00	3	Remove
Edit	Printer	1,000.00	3	Remove

The figure above shows the Fixed Assets sublist. When the **New Fixed Assets** button is clicked, a custom (child) Fixed Assets record opens. After adding data to the Fixed Assets record and saving it, the record will appear as a sublist line item.

The following figure shows that two Fixed Assets (child) records have been added as sublist line items to the (parent) customer record for Adam Fitzpatrick.

The screenshot shows the 'Customer' record edit screen for 'Abe Simpson'. At the top, there are buttons for Save, Cancel, Reset, Merge, and Actions. Below that is the 'Primary Information' section with fields for CUSTOM FORM (set to 'Custom Customer Form 4'), PARENT COMPANY (dropdown), STATUS (set to 'CUSTOMER-Closed Won'), SALES REP (set to 'Jon Baker'), and WEB ADDRESS. To the right are fields for CATEGORY (set to 'From advertisement') and COMMENTS. Under the 'Email | Phone | Address' section, there are fields for EMAIL (set to 'asimpson@netsuite.com'), ALT. PHONE (set to '555-1234'), and PHONE (set to '504-231-1111'). Below these sections is a 'LAST SALES ACTIVITY' summary. The main content area has tabs for Relationships, Communication, Address, Sales, Marketing, Support, Financial, Preferences, System Information, Custom, and Fixed Assets. The 'Fixed Assets' tab is selected, showing a list of assets with columns for NAME, COST, USEFUL LIFE IN YEARS, and REMOVE. The assets listed are: '2003 Ford Expedition' (cost 32,885.00, useful life 5 years, remove), 'Dell Dimension XPS' (cost 1,649.00, useful life 3 years, remove), and 'Printer' (cost 1,000.00, useful life 3 years, remove).

## Understanding Custom Child Record Sublist IDs

Unlike other sublists, there are no standard IDs that can be documented for custom child record sublists. The internal ID for the sublist itself, as well as for all of its associated fields, will be unique to each custom child record sublist.

See these topics for guidelines on determining which IDs to reference in [Sublist APIs](#):

- Determining the Sublist ID
- Determining Field IDs

## Determining the Sublist ID

The internal ID for a custom child record sublist is `recmach + field_id_for_the_parent_field` (for example: `recmachcustrecord112`).

When using [Sublist APIs](#) the value of the `type` parameter in nlapi functions (or the `group` parameter in nlobjRecord sublist-related methods) will look something like the following:

```
nlapiGetLineItemValue('recmachcustrecord102', fldnam, linenum )
```

The following steps describe where to look in NetSuite to get the internal ID of a custom child record sublist.

To get the internal ID of a custom child record sublist (the field ID for the parent record):

1. Go to the record definition for the custom record type (see figure).
2. On the **Fields** tab > **ID** column, notice the internal IDs for two different List/Record field types.

The field definition for the field called **New Customer** (internal ID: **custrecord102**) shows that it is the parent field for the Fixed Assets records that appear as children on Customer records. Hence, the internal ID for the custom child Fixed Assets sublist on Customer records will be **recmachcustrecord102**.

DESCRIPTION	ID	TYPE	LIST/RECORD	TAB	SHOW IN LIST
Cost	custrecord1	Currency			No
Salvage Value	custrecord2	Currency			No
Depreciable Cost	custrecord3	Currency			No
Useful Life in Years	custrecord4	Decimal Number			No
Description	custrecord5	Text Area			No
<b>New Customer</b>	<b>custrecord111</b>	List/Record	<b>Customer</b>		Yes

An alternative approach for obtaining the internal ID of the parent field is by doing the following:

1. On the custom child record sublist, click the button to create a new child record (see figure for a general example).

The screenshot shows a NetSuite Customer record for 'Abe Simpson'. The 'Fixed Assets' tab is selected. A red box highlights the 'New Fixed Assets' sublist, which contains the following data:

EDIT	NAME	COST	USEFUL LIFE IN YEARS	REMOVE
Edit	2003 Ford Expedition	32,885.00	5	<a href="#">Remove</a>
Edit	Dell Dimension XPS	1,649.00	3	<a href="#">Remove</a>
Edit	Printer	1,000.00	3	<a href="#">Remove</a>

- When the new child record opens (see figure below), notice the field that ties the child record back to the parent record.

In this case, the **New Customer** field shows that the customer record for Abe Simpson is the parent of the Fixed Asset record. The field level help popup window for **New Customer** lists the field's internal ID as `custrecord102`. Therefore, the internal ID for the Fixed Asset sublist appearing on the (parent) customer record is `recmachcustrecord102`.

The screenshot shows a NetSuite Fixed Assets record. A field level help window is open over the 'NEW CUSTOMER' dropdown, displaying the following information:

This is a custom field of type List/Record  
Field ID: custrecord111

- Click the back button in your browser to navigate away the new record if you do not want to enter data.



**Note:** Internal IDs for custom child record sublists also appear in the SuiteScript Debugger when you load the record that includes the sublist.

## Determining Field IDs

Use these steps to get internal field IDs on a custom child record sublist:

1. Go to the record definition for the custom record (for example, go to Customization > Lists, Records, & Fields > Record Types and select your custom record in the **Record Types** list).
2. On the Field tab of the Custom Record Type page (see figure), all field internal IDs appear in the ID column. These are the IDs you will reference as the *fldnam* value in all **Sublist APIs**.

DESCRIPTION	ID	TYPE	LIST/RECORD	TAB	SHOW IN LIST
Cost	custrecord1	Currency			No
Salvage Value	custrecord2	Currency			No
Depreciable Cost	custrecord3	Currency			No
Useful Life in Years	custrecord4	Decimal Number			No
Description	custrecord5	Text Area			No
New Customer	custrecord111	List/Record	Customer		Yes

### Example:

```
//Get the value of the Cost field on the first line (see the following figure)
nlapiGetLineItemValue('recmachcustrecord102', 'custrecord1', 1)
```

Note that you can also get/set values for fields that do not appear in the sublist UI. For example, the following line sets the value of the **Salvage Value** field in the record Fixed asset 3 (see figure). The internal ID for Salvage Value is custrecord2. See this value on the Custom Record Type definition page for the Fixed Assets record type.

```
nlapiSetCurrentLineItemValue('recmachcustrecord102', 'custrecord2', 700);
```

The screenshot shows a NetSuite customer record edit screen for 'Abe Simpson'. The 'Fixed Assets' tab is selected. A sublist titled 'Fixed Assets' is displayed, containing three items:

	NAME	COST	USEFUL LIFE IN YEARS	REMOVE
Edit	2003 Ford Expedition	32,885.00	5	<a href="#">Remove</a>
Edit	Dell Dimension XPS	1,649.00	3	<a href="#">Remove</a>
Edit	Printer	1,000.00	3	<a href="#">Remove</a>

You can get or set values for fields that appear in the Fixed Assets sublist. You can also set or get values that exist on the record, but do not appear in the sublist.

## Scripting with Custom Child Record Sublists

Custom child record sublists are inline editor sublists. Consequently, they support all standard [Sublist APIs](#) that run on other inline editor sublists. A custom child record sublist is unique only because it is not identified by a standard sublist internal ID, nor does it contain a standard set of field IDs. Otherwise, like all other inline editor sublists, you can add and remove line items; you can get and set values on existing line items; and the first line number (linenum) for all sublists is 1, not 0.



**Important:** Be aware that you cannot execute validate line functions on custom child record sublists. Validate line functions are executed when a client event occurs prior to a line being added to a sublist.

## Adding a record to a custom child record sublist

The following snippet shows how to add a new Fixed Assets record to the Fixed Assets sublist. This is a server-side script in which the customer record object is loaded into the system, and the Fixed Assets sublist (recmachcustrecord102) is being accessed through methods on the `nlobjRecord` object.

```
var rec = nlapiLoadRecord('customer', 142);
//Call the nlobjRecord selectNewLineItem method to add a new line.
```

```
//Note: Call selectLineItem(...) if the line already exists and you are just updating it.
rec.selectNewLineItem('recmachcustrecord102');
//Set the value for the record name
rec.setCurrentLineItemValue('recmachcustrecord102', 'name', 'Printer');
//Set the value for the Cost field
rec.setCurrentLineItemValue('recmachcustrecord102', 'custrecord1', 1000);
//Set the value for the Useful Life in Years field
rec.setCurrentLineItemValue('recmachcustrecord102', 'custrecord4', '3');
//Commit your sublist changes
rec.commitLineItem('recmachcustrecord102');
//Submit the updated Customer record
var id = nlapiSubmitRecord(rec, true);
```

The screenshot shows a NetSuite Customer Record for 'Abe Simpson'. The 'Primary Information' section includes fields like CUSTOM FORM (Custom Customer Form 4), PARENT COMPANY (<Type then tab>), CATEGORY (From advertisement), and STATUS (CUSTOMER-Closed Won). The 'Email | Phone | Address' section contains EMAIL (asimpson@netsuite.com), ALT. PHONE (555-1234), and PHONE (504-231-1111). The 'LAST SALES ACTIVITY' section is empty. Below the main record details, there is a tab bar with 'Relationships', 'Communication', 'Address', 'Sales', 'Marketing', 'Support', 'Financial', 'Preferences', 'System Information', 'Custom', and 'Fixed Assets'. The 'Fixed Assets' tab is selected, displaying a sublist titled 'Fixed Assets'. The sublist has columns for EDIT, NAME, COST, USEFUL LIFE IN YEARS, and REMOVE. It lists three assets: '2003 Ford Expedition' (cost 32,885.00, useful life 5 years), 'Dell Dimension XPS' (cost 1,649.00, useful life 3 years), and 'Printer' (cost 1,000.00, useful life 3 years). The 'Printer' row is highlighted with a red border.

## Deductions Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **deduction**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Deductions sublist appears on the [Paycheck Journal](#) record and the Employee record. To see the internal IDs associated with the Deductions sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Demand Plan Detail Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **demandplandetail**.

The Demand Plan Detail sublist appears on the [Item Demand Plan](#) record type. This record stores the expected future demand for an item, based on previous or projected demand. This record's body fields uniquely identify a demand plan by item, subsidiary (OneWorld accounts), location (Multi-Location Inventory accounts), start date, and end date. Another body field, **demandplancalendarype**, determines the interval to be used for demand plan quantity values, either monthly, weekly, or daily.

The Demand Plan Detail sublist is a matrix that is similar to the [Pricing Sublist](#). This matrix stores projected quantities demanded by date. Each row in the matrix represents a specific month, week, or day, and each column in the matrix represents an expected quantity demand.

Functionally, this sublist shares many of the characteristics of List Sublists. However, scripting with the Demand Plan Detail sublist is not like scripting with most other sublists in NetSuite. You must use [Matrix APIs for the Demand Plan Detail Sublist](#) to access quantity values on a per-row, per-column basis, similar to the way that item pricing values are accessed. These APIs are a subset of the [Sublist APIs](#) more commonly used for scripting with other sublists.

The format of the Demand Plan Detail sublist depends on the values set in body fields for the start date of the plan, the end date of the plan, and the time period to be used (monthly, weekly, or daily). Because of this dependence, it is recommended that you work with the Item Demand Plan record and the Demand Plan Detail sublist in dynamic mode. See the help topic [Working with Records in Dynamic Mode](#).

Be aware of the following requirements:

- To script with the Item Demand Plan record and the Demand Plan Detail sublist for inventory items, the Demand Planning feature must be enabled. For assembly/BOM items, the Work Orders feature also must be enabled.
- Demand plans are supported only for item(s) that have the **supplyreplenishmentmethod** field set to Time Phased.
- Required body field values must be defined before matrix field values can be edited. In dynamic mode, current values may be retrieved. Start date and end date body fields default to the first day and last day of the current year.

For more details and code samples for each type of demand plan, see the following:

- [Monthly Demand Plan](#)
- [Weekly Demand Plan](#)
- [Daily Demand Plan](#)

### Monthly Demand Plan

A monthly demand plan includes a row for each month within the body field start date and end date, and one quantity column for each month.

**Manual Item Demand Plan**

**Primary Information**

<b>SUBSIDIARY *</b>	<b>UNIT OF MEASURE</b>			
<input type="text"/>	<input type="text"/>			
<b>LOCATION *</b>	<b>MEMO</b>			
<input type="text"/>	<input type="text"/>			
<b>ITEM *</b>				
<input type="text"/>				
<b>YEAR</b>	<b>MONTH</b>	<b>START DATE</b>	<b>END DATE</b>	<b>VIEW</b>
2015		1/1/2015	12/31/2015	Monthly

START DATE	END DATE	MONTHLY CALCULATED	QUANTITY
1/1/2015	1/31/2015		
2/1/2015	2/28/2015		
3/1/2015	3/31/2015		
4/1/2015	4/30/2015		
5/1/2015	5/31/2015		
6/1/2015	6/30/2015		
7/1/2015	7/31/2015		
8/1/2015	8/31/2015		
9/1/2015	9/30/2015		
10/1/2015	10/31/2015		
11/1/2015	11/30/2015		
12/1/2015	12/31/2015		

- The sublist startdate and enddate fields are system-calculated and read-only.
  - The startdate is the date of the first day of the month represented by each row.
  - The enddate is the date of the last day of the month represented by each row.
  - The month for row 1 is the month set in the body field start date, the month for row 2 is the next month, and so on, until the month set in the body field end date is reached.
- The values for the quantity field can be set in SuiteScript. For monthly demand plans, the column parameter for this field is always 1.

## Monthly Demand Plan Code Sample

The following code sets quantities for the months of January and February, 2011:

```
var record = nlapiCreateRecord('itemdemandplan', {recordmode: 'dynamic'});
record.setFieldValue('demandplancalendartype','MONTHLY');

record.setFieldValue('subsidiary', 1);
record.setFieldValue('location', 1);
record.setFieldValue('item', 165);
record.setFieldValue('startdate','1/1/2015');
record.setFieldValue('enddate','12/31/2015');

record.selectLineItem('demandplandetail', '1');
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 100);
```

```
record.selectLineItem('demandplandetail', '2');
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 200);

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

## Weekly Demand Plan

A weekly demand plan includes a row for each week contained in the time period set by the body field start date and end date, and one quantity column for each week.

START DATE	END DATE	WEEKLY CALCULATED	QUANTITY
12/28/2014	1/3/2015		
1/4/2015	1/10/2015		
1/11/2015	1/17/2015		
1/18/2015	1/24/2015		
1/25/2015	1/31/2015		
2/1/2015	2/7/2015		
2/8/2015	2/14/2015		
2/15/2015	2/21/2015		
2/22/2015	2/28/2015		
3/1/2015	3/7/2015		
3/8/2015	3/14/2015		
3/15/2015	3/21/2015		
3/22/2015	3/28/2015		
3/29/2015	4/4/2015		

- The sublist startdate and enddate fields are system-calculated and read-only.
  - The startdate is the date of the first day of the week represented by each row.
  - The enddate is the date of the last day of the week represented by each row.



**Note:** The first day of the week by default is Sunday, but may vary according to the company preference set for First Day of the Week at Setup > Company > General Preferences.

- The week for row 1 is the week of the date set in the body field start date. Note that unless the body field start date happens to be the first day of the week, the startdate for this first row may precede the body field start date.

- The week for the final sublist row is the week of the date set in the body field end date. Note that unless the body field end date happens to be the last day of the week, the enddate for this last row may be after the body field enddate.
- The values for the quantity field can be set in SuiteScript. For weekly demand plans, the column parameter for this field is always 1.

## Weekly Demand Plan Code Sample

The following code sets quantities for the first two weeks of 2011:

```
var record = nlapiCreateRecord('itemdemandplan',{recordmode: 'dynamic'});  
record.setFieldValue('demandplancalendartype',WEEKLY);  
  
record.setFieldValue('subsidiary', 1);  
record.setFieldValue('location', 1);  
record.setFieldValue('item', 165);  
record.setFieldValue('startdate','1/1/2015');  
record.setFieldValue('enddate','12/31/2015');  
  
record.selectLineItem('demandplandetail', '1');  
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 100);  
  
record.selectLineItem('demandplandetail', '2');  
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 200);  
  
var id = nlapiSubmitRecord(record,true);
```

## Daily Demand Plan

A daily demand plan includes a row for each week contained in the time period set by the body field start date and end date, and seven quantity columns for each week, one for each day of the week.

**Manual Item Demand Plan**

[Save](#) [Cancel](#) [Reset](#)

**Primary Information**

SUBSIDIARY *	UNIT OF MEASURE								
LOCATION *	MEMO								
ITEM *									
YEAR 2015	MONTH 1	START DATE 1/1/2015	END DATE 12/31/2015	VIEW Daily					
START DATE	END DATE	WEEKLY CALCULATED	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
12/28/2014	1/3/2015								
1/4/2015	1/10/2015								
1/11/2015	1/17/2015								
1/18/2015	1/24/2015								
1/25/2015	1/31/2015								
2/1/2015	2/7/2015								
2/8/2015	2/14/2015								
2/15/2015	2/21/2015								
2/22/2015	2/28/2015								
3/1/2015	3/7/2015								
3/8/2015	3/14/2015								
3/15/2015	3/21/2015								
3/22/2015	3/28/2015								
3/29/2015	4/4/2015								

- The sublist startdate and enddate fields are system-calculated and read-only.
    - The startdate is the date of the first day of the week represented by each row.
    - The enddate is the date of the last day of the week represented by each row.
- Note:** The first day of the week by default is Sunday, but may vary according to the company preference set for First Day of the Week at Setup > Company > General Preferences.
- The week for row 1 is the week of the date set in the body field start date. Note that unless the body field start date happens to be the first day of the week, the startdate for this first row may precede the body field start date.
  - The week for the final sublist row is the week of the date set in the body field end date. Note that unless the body field end date happens to be the last day of the week, the enddate for this last row may be after the body field enddate.
  - The values for the quantity fields can be set in SuiteScript.
    - The column parameter for a quantity field is 1,2,3,4,5,6, or 7, depending upon the day of the week.
    - In the screenshot above, the week starts with Sunday, which is the default first day of the week, and in this case, maps to a column parameter of 1. However, 1 does not always map to Sunday; it maps to the first day of the week as set in the company preferences.

## Daily Demand Plan Code Sample

```

var record = nlapiCreateRecord( 'itemdemandplan',{recordmode: 'dynamic'});
record.setFieldValue('demandplancalendartype','DAILY');

record.setFieldValue('subsidiary', 1);
record.setFieldValue('location', 1);
record.setFieldValue('item', 165);
record.setFieldValue('startdate','1/1/2011');
record.setFieldValue('enddate','12/31/2011');

record.selectLineItem('demandplandetail', '1'); // week of 12/26/2010 to 1/1/2011
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 100); //sunday
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '2', 101); //monday
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '3', 102); //tuesday

record.selectLineItem('demandplandetail', '2'); //week of 1/2/2011 to 1/8/2011
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '1', 200); //sunday
record.setCurrentLineItemMatrixValue('demandplandetail', 'quantity', '5', 200); //thursday

var id = nlapiSubmitRecord(record,true);

```

## Matrix APIs for the Demand Plan Detail Sublist

Use the following matrix APIs with the Demand Plan Detail sublist:

- [nlapiGetCurrentLineItemMatrixValue\(type, fldnam, column\)](#)
- [nlapiSetCurrentLineItemMatrixValue\(type, fldnam, column, value, firefieldchanged, synchronous\)](#)



**Note:** With the two APIs above, use this API first to select an existing line:  
[nlapiSelectLineItem\(type, linenum\).](#)

- [nlapiGetLineItemMatrixField\(type, fldnam, linenum, column\)](#)
- [nlapiGetLineItemMatrixValue\(type, fldnam, linenum, column\)](#)
- [nlapiFindLineItemMatrixValue\(type, fldnam, val, column\)](#)

For more information about APIs, see [Sublist APIs](#).

## Deposits Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **deposit**. The sublist is a **list** sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Deposits sublist appears on the Customer Payment and Customer Refund records. To see the internal IDs associated with the Deposits sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Direct Mail Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **campaigndirectmail**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Direct Mail sublist appears on the campaign record. To see the internal IDs associated with the Direct Mail sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Download Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **download**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Download sublist appears on the customer record. To see the internal IDs associated with the Download sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Earnings Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **earning**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Earnings sublist appears on the [Paycheck Journal](#) record and the Employee record. To see the internal IDs associated with the Earning sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## E-mail Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **campaignemail**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The E-mail sublist appears on the campaign record. To see the internal IDs associated with the E-mail sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Employee Taxes Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **employeetax**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Employee Taxes sublist appears on the [Paycheck Journal](#) record. To see the internal IDs associated with the Employee Tax sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Escalate To Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **escalateto**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Escalate To sublist appears on the support case record. To see the internal IDs associated with the Escalate sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Expenses Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **expense**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Expenses sublist appears on the following records: Check, Vendor Bill, Purchase Order, Item Receipt, Expense Report. To see the internal IDs associated with the Expenses sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Group Pricing Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **grouppricing**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Group Pricing sublist appears on the Customer record. To see the internal IDs associated with the Group Pricing sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Item Fulfillment/Receipt Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **item**. This is a **list** sublist. (In the NetSuite Help Center, see [List Sublists](#) for information on this sublist type.)

The Item Fulfillment/Item Receipt sublist appears on the Item Fulfillment and Item Receipt records. To see the internal IDs associated with the Item Fulfillment/Item Receipt sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.



**Important:** There are two types of item sublists in NetSuite. Although both item sublists have the same internal ID (`item`), the sublists themselves appear on different record types. Also note that the item sublist referenced in this section is a list sublist. The item sublist referenced in the section [Items Sublist](#) is an inline editor sublist type.

## Items Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is `item`. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Items appears on the following records: Cash Refund, Cash Sale, Check, Credit Memo, Estimate/Quote, Invoice, Opportunity, Purchase Order, Return Authorization, Sales Order, Vendor Bill, Work Orders, and Transfer Order. To see the internal IDs associated with the Items sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.



**Important:** There are two types of item sublists in NetSuite. Although both item sublists have the same internal ID (`item`), the sublists themselves appear on different record types. Also note that the item sublist referenced in this section is an [inline editor](#) sublist type. The item sublist referenced in the section [Item Fulfillment/Receipt Sublist](#) is a [list](#) sublist type.

## Usage Notes

The following table provides usage notes for specific sublist fields on this sublist.

Sublist Field Internal ID	Sublist Field UI Label	Note
<code>altsalesamt</code>	Alt. Sales Amount	This sublist field is NOT scriptable on the Cash Sale and Invoice records.
<code>costestimatetype</code>	Cost Estimate Type	This field supports both client and server scripting. It is also available in search.
<code>costestimate</code>	Est. Extended Cost	This field supports both client and server scripting. It is also available in search.
<code>estgrossprofit</code>	Est. Gross Profit	This field cannot be scripted with client/server SuiteScript. It is, however, available in search.

Sublist Field Internal ID	Sublist Field UI Label	Note
estgrossprofitpercent	Est. Gross Profit Percent	This field cannot be scripted with client/server SuiteScript. It is, however, available in search.

## Item Pricing Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **itempricing**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Item Pricing sublist appears on the customer record. To see the internal IDs associated with the Item Pricing sublist, refer to the [SuiteScript Records Browser](#).

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Lead Nurturing Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **campaigndrip**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Lead Nurturing sublist appears on the campaign record. To see the internal IDs associated with the Lead Nurturing sublist, refer to the [SuiteScript Records Browser](#).

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Line Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **line**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Line sublist appears on the Journal Entry record and on the Intercompany Journal Entry record. To see the internal IDs associated with the Line sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Members Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **member**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Members sublist appears on the Assembly Item, Lot Numbered Assembly Item, and Kit Item records. To see the internal IDs associated with the Members sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The following table provides usage notes for specific sublist fields on this sublist.

Sublist Field Internal ID	Sublist Field UI Label	Note
taxschedule	Scheduled	This sublist field is visible only in the UI when the Advanced Taxes feature is enabled.

## Orders Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **order**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Order sublist appears on the [Item Supply Plan](#) record. To see the internal IDs associated with the Order sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Other Events Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **campaignevent**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Other Events sublist appears on the campaign record. To see the internal IDs associated with the Other Events sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Partners Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **partners**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Partners sublist appears on the following records: Opportunity, Sales Order, Invoice, Cash Sale, Estimate, Cash Refund, Return Authorization, Credit Memo, Work Order, Lead, Prospect, Customer. To see the internal IDs associated with the Partners sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

The Multi-Partner Management feature must be enabled in your account for this sublist to appear.

## Pricing Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The Pricing sublist is often referred to as a **pricing matrix**, since there can be multiple prices specified for an item, as determined by one or more price levels and one or more quantity levels.

Functionally, the Pricing sublist shares many of the characteristics of [List Sublists](#). However, scripting to the Pricing sublist is not like scripting to other sublists in NetSuite. For this reason it is recommended that you read all of the following topics to learn about using SuiteScript on this sublist. These topics do not need to be read in order, although it is recommended:

- [What is the Pricing Matrix?](#)
- [Pricing Sublist Feature Dependencies](#)
- [Pricing Sublist Internal IDs](#)
- [Pricing Sublist Code Sample](#)
- [Matrix Sublist APIs and Standard Sublist APIs](#)





**Note:** The screenshots in this section depict the NetSuite user interface prior to Version 2010 Release 2.

For general information on item pricing, see these topics in the NetSuite Help Center:

- [Item Pricing](#)
- [Setting Up Item Pricing](#)
- [Using Item Records](#)

To see which records the Pricing sublist appears on, see [Records that Include the Pricing Sublist](#).

## What is the Pricing Matrix?

Depending on the features enabled in your account, the Pricing sublist on many item records can resemble a **matrix** of rows and columns of various prices (see figure). To access the price values on a per-row, per-column basis, you must use [Matrix APIs](#). These APIs are a subset of the non-matrix [Sublist APIs](#) that are more commonly used when scripting with other sublists.



**Note:** Non-matrix sublist APIs can also be used on the Pricing sublist. However, they are not used to get|set values considered to be part of the **pricing matrix**. For information on when to use matrix and non-matrix APIs on the Pricing sublist, see [Matrix Sublist APIs and Standard Sublist APIs](#).

The figure below provides an overview of the rows and columns considered to be the **pricing matrix**. As previously stated, the configuration of the Pricing sublist greatly depends on the features enabled in your account. However, regardless of the features set, all configurations will have some variation of a row / column matrix layout like the one shown below.

The screenshot shows the NetSuite Pricing sublist interface. At the top, there are dropdown menus for 'QUANTITY PRICING SCHEDULE' (set to 'Pricing Schedule 1'), 'CALCULATE QUANTITY DISCOUNTS' (set to 'By Line Quantity'), and 'PRICING GROUP'. A checked checkbox labeled 'USE MARGINAL RATES' is present. Below these settings, there is a row of currency codes: 'USA • British pound • Canadian Dollar • Euro • Yen •'. The main area is a grid table with columns for 'PRICE LEVEL' (Default Discount %, Base Price, Alpha price level, Alternate Price 1, Alternate Price 2, Alternate Price 3, Alternate Price 4, Beta price Level, Wholesale Price, Online Price) and rows for 'QTY 0', 'QTY 100', 'QTY 200', 'QTY 300', and 'QTY 400'. The grid is highlighted with a red border.

PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY 400
Default Discount %			0.0%	5.0%	10.0%	
Base Price		50.00	50.00	52.50	55.00	
Alpha price level		75.00	75.00	78.75	82.50	
Alternate Price 1						
Alternate Price 2						
Alternate Price 3						
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50	
Beta price Level	50.0%	75.00	75.00	78.75	82.50	
Wholesale Price						
Online Price						

This code snippet shows the kind of values you will typically set when working with price values in the pricing matrix. The internal ID of the Pricing sublist, as well as its field IDs, will change depending on the features enabled in your account.



**Note:** See [Pricing Sublist Feature Dependencies](#) and [Pricing Sublist Internal IDs](#) for more information.

### Example

```
//nlapiGetLineItemMatrixValue(type, fldnam, linenum, column)
nlapiGetLineItemMatrixValue('price', 'price', 2, 1);
```

In this sample you:

1. Specify the sublist internal ID ( **price** ).
2. Specify the internal ID of the pricing field (which will generally be **price** ).



**Important:** Although the UI labels in this figure show field names such as Alternate Price 1, Alternate Price 2, and Online Price, the internal ID for the **fldnam** parameter is still **price**. The only exception to this is described in [Pricing Sublist Field IDs](#) for the **currency** field.

3. Specify the line number (row) of the price you want to get (in this sample, you are getting the value in row 2 - this is the price for **Alternate Price 1** ).
4. Specify the column number you want to get the value for (in this sample, you are getting the value in column 1 for **Alternate Price 1** ).

	PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY
Default Discount %				0.0%	5.0%	10.0%	
Base Price			50.00	50.00	52.50	55.00	
Alpha price level			75.00	75.00	78.75	82.50	
Alternate Price 1							
Alternate Price 2							
Alternate Price 3							
Alternate Price 4	10.0%		55.00	55.00	57.75	60.50	
Beta price Level	50.0%		75.00	75.00	78.75	82.50	
Wholesale Price							
Online Price							

1	Column one
2	Row two

Price is typically the internal ID for the **fldnam** parameter.

## Pricing Sublist Feature Dependencies

There are three features that, if enabled or disabled in your account, can affect the overall functionality of the Pricing sublist, its appearance in the UI, and the internal IDs that are referenced in SuiteScript.

You can check which of these features are enabled by looking in the UI, or by calling the `nlobjGetContext.getFeature(name)` method and specifying the feature internal ID.

The features that affect the Pricing sublist are:

- [Multiple Currencies](#)

- Multiple Prices
- Quantity Pricing

If none of these features are enabled in your account, then there is no Pricing sublist on the item record, and the field that holds the item price appears on the Basic subtab as **Sales Price** (see figure). The internal ID for this field is **rate**. You do not use [Sublist APIs](#) to set or get values on **rate**. Instead, use [Field APIs](#).

## Multiple Currencies

This feature allows for item prices to be set in multiple currencies. Separate pricing is specified for each currency. On the Pricing sublist you will see subtabs with the name of the currencies specified in your account (see figure).

**Important:** See [Pricing Sublist ID](#) to learn how to determine the internal ID of the Pricing sublist based on whether the Multiple Currencies feature is enabled.

The screenshot shows the Pricing sublist interface. At the top, there are dropdown menus for 'QUANTITY PRICING SCHEDULE' (set to 'Pricing Schedule 1'), 'CALCULATE QUANTITY DISCOUNTS' (set to 'By Line Quantity'), and 'PRICING GROUP'. A checked checkbox labeled 'USE MARGINAL RATES' is also visible. Below these settings, a row of currency tabs is highlighted with a red border: 'USA • British pound • Canadian Dollar • Euro • Yen •'. Underneath this row, there's a table with columns for 'PRICE LEVEL', 'DEFAULT DISCOUNT %', and quantity ranges (QTY 0, 100, 200, 300, etc.). The table lists various price levels like 'Base Price', 'Alpha price level', and 'Alternate Price 1-4', along with their corresponding prices and discounts for each currency.

In the UI, you can check if this feature is enabled by looking at the Pricing sublist itself or by going to Setup > Company > Enable Features. On the Company tab, the **Multiple Currencies** box will be selected if this feature is enabled. Note: Only a NetSuite administrator can enable this feature.

In SuiteScript, you can get the feature status by writing something similar to:

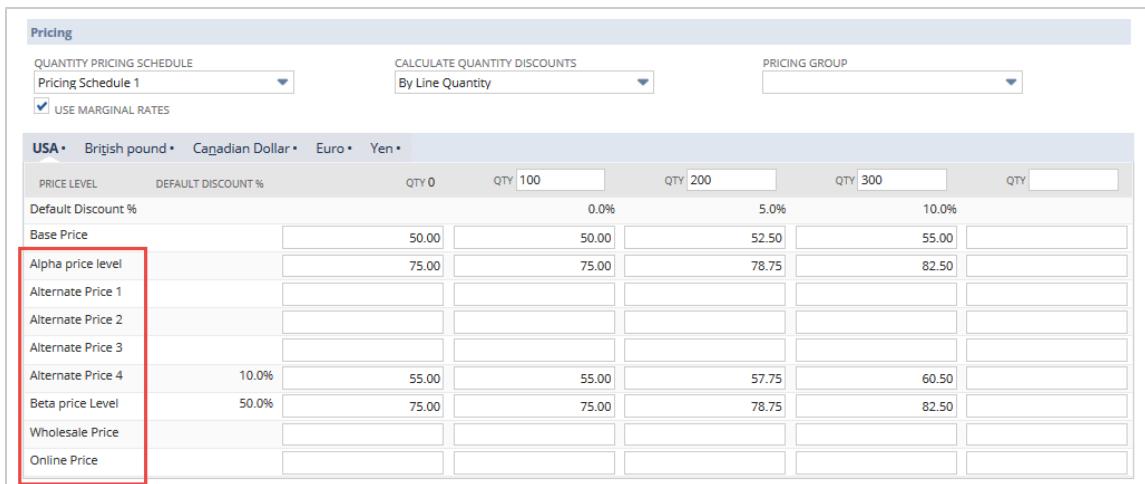
```
var multiCurrency = nlapiGetContext().getFeature('MULTICURRENCY');
```

**Important:** See [Pricing Sublist Code Sample](#) for more details.

## Multiple Prices

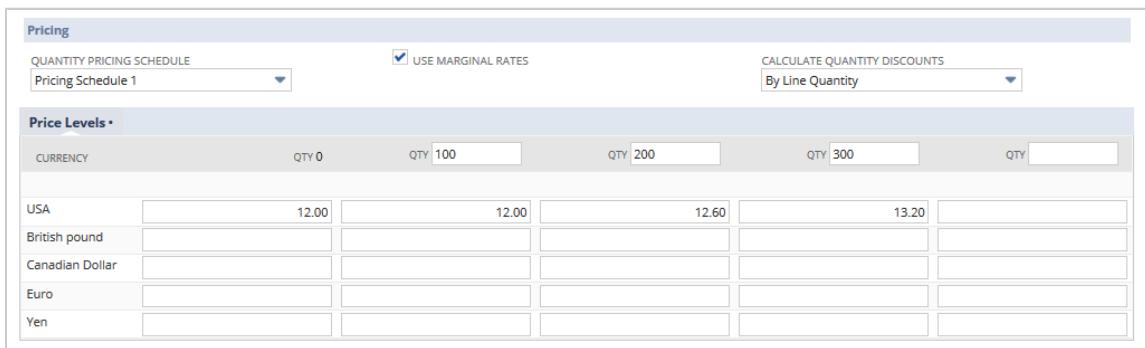
This feature allows different prices to be specified for different conditions or types of customers. This requires that Price Levels are set up. There are a set of standard Price Levels provided by NetSuite, and these can be changed or extended by the customer.

This figure shows the Pricing sublist with the Multiple Prices feature enabled. Notice you can specify multiple prices for the same item.



The screenshot shows the Pricing sublist interface. At the top, there are three dropdown menus: 'QUANTITY PRICING SCHEDULE' (set to 'Pricing Schedule 1'), 'CALCULATE QUANTITY DISCOUNTS' (set to 'By Line Quantity'), and 'PRICING GROUP'. A checked checkbox labeled 'USE MARGINAL RATES' is also present. Below these, there are currency selection buttons for 'USA', 'British pound', 'Canadian Dollar', 'Euro', and 'Yen'. A table displays price levels for different quantities (0, 100, 200, 300, etc.) across various currencies. A red box highlights the first row under 'Base Price', which is labeled 'Alpha price level'. The table shows prices like 50.00, 50.00, 52.50, 55.00 for QTY 0, 100, 200, 300 respectively.

By comparison, this figure shows the Pricing sublist with the Multiple Prices feature disabled. You can set only one price for the item.



The screenshot shows the Pricing sublist interface with the 'Multiple Prices' checkbox unchecked. The rest of the interface is identical to the previous screenshot, including the currency selection buttons and the table displaying price levels for different quantities across various currencies.

In the UI, you can check if this feature is enabled by looking at the Pricing sublist itself or by going to Setup > Company > Enable Features. On the Transactions subtab, the **Multiple Prices** check box will be selected if this feature is enabled. Note: Only a NetSuite administrator can enable this feature.

In SuiteScript, you can get the feature status by writing something similar to:

```
var multiPrice = nlapiGetContext().getFeature('MULTPRICE');
```



**Important:** See [Pricing Sublist Code Sample](#) for more details.

## Quantity Pricing

This feature allows the item price to vary based on the quantity of items sold. Specifically, this feature allows different quantity levels to be specified and allows the price to vary at each quantity level.

This figure shows the Pricing sublist with the Quantity Pricing feature enabled.



**Note:** When the Quantity Pricing feature is enabled, an administrator can specify the number of Qty columns that appear on the Pricing sublist. The following figure shows that four Qty columns have been specified. Set the Qty preference by going to Setup > Accounting > Accounting Preferences > Items & Transactions. In the **Maximum # of Quantity-based Price Levels** field, specify the number of columns.

Pricing		QUANTITY PRICING SCHEDULE		CALCULATE QUANTITY DISCOUNTS		PRICING GROUP	
		Pricing Schedule 1		By Line Quantity			
		<input checked="" type="checkbox"/> USE MARGINAL RATES					
<b>USA • British pound • Canadian Dollar • Euro • Yen •</b>							
PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY 400	QTY 500
Default Discount %			0.0%	5.0%	10.0%		
Base Price		50.00	50.00	52.50	55.00		
Alpha price level		75.00	75.00	78.75	82.50		
Alternate Price 1							
Alternate Price 2							
Alternate Price 3							
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50		
Beta price Level	50.0%	75.00	75.00	78.75	82.50		
Wholesale Price							
Online Price							

Item pricing is determined by values specified in the Qty fields.

By comparison, this figure shows the Pricing sublist with the Quantity Pricing feature disabled. Item prices are not determined by the quantities specified.

No Qty fields exist in the following image:

Pricing		
PRICING GROUP		
PRICE LEVEL	DEFAULT DISCOUNT %	AMOUNT
Base Price		100.00
Alpha price level		
Alternate Price 1		
Alternate Price 2		
Alternate Price 3		
Alternate Price 4	10.0%	110.00
Beta price Level	50.0%	150.00
Wholesale Price		
Online Price		

In the UI, you can check if this feature is enabled by looking at the Pricing sublist itself or by going to Setup > Company > Enable Features. On the Transactions subtab, the **Quantity Pricing** check box will be selected if this feature is enabled. Note: Only a NetSuite administrator can enable this feature.

In SuiteScript, you can get the feature status by writing something similar to:

```
var quantityPricing = nlapiGetContext().getFeature('QUANTITYPRICING');
```



**Important:** See [Pricing Sublist Code Sample](#) for more details.

## Pricing Sublist Internal IDs

As discussed in [Pricing Sublist Feature Dependencies](#), the Pricing sublist looks and functions different depending on the features set in your account.

See [Pricing Sublist ID](#) for the internal ID of the Pricing sublist depending on features enabled in your account.

See [Pricing Sublist Field IDs](#) for all other field IDs associated with this sublist.

## Pricing Sublist ID

In SuiteScript, the internal ID of the Pricing sublist is determined by the features enabled in your NetSuite account.

If the Multiple Currencies feature is **not** enabled in your account, the internal ID for the Pricing sublist is **price**. This means that you will set the **type** parameter in APIs such as `nlapiGetMatrixField( type, fldnam, column)` and `nlapiSetLineItemMatrixValue( type, fldnam, linenum, column, value)` to **price**.

If Multiple Currencies is enabled, then there are separate Pricing sublists per currency (see figure).

Pricing		QUANTITY PRICING SCHEDULE		CALCULATE QUANTITY DISCOUNTS		PRICING GROUP	
		Pricing Schedule 1		By Line Quantity			
<input checked="" type="checkbox"/> USE MARGINAL RATES							
<b>USA • British pound • Canadian Dollar • Euro • Yen •</b>							
PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY	
Default Discount %			0.0%	5.0%	10.0%		
Base Price		50.00	50.00	52.50	55.00		
Alpha price level		75.00	75.00	78.75	82.50		
Alternate Price 1							
Alternate Price 2							
Alternate Price 3							
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50		
Beta price Level	50.0%	75.00	75.00	78.75	82.50		
Wholesale Price							
Online Price							

Each currency pricing list will have its own internal ID. For example, the internal ID for the currency called USA will be **price1**. This ID reflects the internal ID of the sublist (**price**) and the internal ID of the USA currency (**1**).

The internal ID for the Canadian dollar sublist will be **price3**. This reflects the internal ID of the sublist (**price**) and the internal ID of the Canadian dollar currency (**3**).

This figure shows the currencies that have been set in this account. Notice the Internal ID for each currency is the numeric value appended to **price**. When the Multiple Currencies feature is enabled, you can see the internal ID for each currency by going to Lists > Accounting > Currencies.

Currencies					
		New	Refresh		
		SHOW INACTIVES			
INTERNAL ID	NAME	ISO CODE	BASE CURRENCY	INACTIVE	AUTOMATIC UPDATE
1	USA	USD	Yes	No	Yes
2	British pound	GBP	No	No	Yes
3	Canadian Dollar	CAD	No	No	Yes
4	Euro	EUR	No	No	Yes
5	Yen	JPY	No	No	Yes

Based on the internal ID in the figure above, you will set the type parameter in APIs such as nlapiGetMatrixField as follows:

```
nlapiGetMatrixField(price1, fldnam, column) // if scripting on the USA tab  
nlapiGetMatrixField(price2, fldnam, column) // if scripting on the British pound tab  
nlapiGetMatrixField(price3, fldnam, column) // if scripting on the Canadian dollar tab  
nlapiGetMatrixField(price4, fldnam, column) // if scripting on the Euro tab
```

For topics related to this one, see [Pricing Sublist Feature Dependencies](#).

## Pricing Sublist Field IDs

This table provides the internal IDs for all fields associated with the Pricing sublist. Field types are categorized as **matrix** fields, **sublist** fields, and **body** fields.

In SuiteScript, use the IDs that appear in the “Field Internal ID” column for the **fldnam** values in [Sublist APIs](#) and [Field APIs](#).

Field UI Label	Field Internal ID	Field Type	Mandatory	Field Notes
<b>Matrix Fields</b>				
Base Price Qty	price	string	true	The price for that level and quantity. See <a href="#">Figure 1 - Matrix Fields</a> .
<div style="border: 1px solid #ccc; padding: 10px;"> <b>Important:</b> When using matrix APIs (ie., any API that has the word <b>Matrix</b> in its name), <b>price</b> will generally be the value specified for the <b>fldnam</b> parameter. The exception to this is if the Multiple Currencies feature is enabled in your account.         </div>				
Default Discount %	discount			See <a href="#">Figure 2 - Sublist Fields</a> . See also <a href="#">Standard Sublist APIs</a> for a code sample that references this internal ID.
	currency			The currency field is a hidden field. It is not visible in the UI. See <a href="#">Figure 2 - Sublist Fields</a> . This field is scriptable only when the Multiple Currencies feature is enabled. The internal IDs for the <b>fldname</b> parameter in matrix APIs will be

Field UI Label	Field Internal ID	Field Type	Mandatory	Field Notes
Price Level	pricelevel			price1currency, price2currency, pricecurrency3, etc., to reflect the currency internal ID.  The pricelevel for this price. See <a href="#">Figure 2 - Sublist Fields</a> . See also <a href="#">Standard Sublist APIs</a> for a code sample that references this internal ID.
<b>Body Fields</b>				
Quantity Pricing Schedule	quantitypricingschedule	select	false	If a quantity pricing schedule has been specified in the UI drop-down, item prices will be calculated for the pricing matrix according to the specified schedule. See <a href="#">Figure 3 - Body Fields</a> .  Also note that the value of the quantity pricing schedule sets the value of the Calculate Quantity Discounts (overallquantitypricingty pe) field. Use <a href="#">Field APIs</a> to access this field.
Calculate Quantity Discounts	overallquantitypricingty pe	select	false	Used to determine the quantity amount at the time the item is priced on the order. (This field does not change price settings in the matrix). See <a href="#">Figure 3 - Body Fields</a> . Use <a href="#">Field APIs</a> to access this field.
Use Marginal Rates	usemarginalrates	checkbox	false	Used to determine how the quantity discounts are applied at the time the item is priced on the order with a specified quantity. (This field does not change the price settings in the matrix.) See <a href="#">Figure 3 - Body Fields</a> . Use <a href="#">Field APIs</a> to access this field.
Pricing Group	pricinggroup	select	false	Used to provide customer-specific pricing. Could affect the pricing at the time the item is placed on the order and associated with a specific customer. See <a href="#">Figure 3 - Body Fields</a> . Use <a href="#">Field APIs</a> to access this field.

**Figure 5. Figure 1 - Matrix Fields**

Pricing		QUANTITY PRICING SCHEDULE		CALCULATE QUANTITY DISCOUNTS		PRICING GROUP	
		Pricing Schedule 1		By Line Quantity			
<input checked="" type="checkbox"/> USE MARGINAL RATES							
PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY	
Default Discount %			0.0%	5.0%	10.0%		
Base Price		50.00	50.00	52.50	55.00		
Alpha price level		75.00	75.00	78.75	82.50		
Alternate Price 1							
Alternate Price 2							
Alternate Price 3							
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50		
Beta price Level	50.0%	75.00	75.00	78.75	82.50		
Wholesale Price							
Online Price							

Pricing sublist matrix field.

**Figure 6. Figure 2 - Sublist Fields**

Pricing		QUANTITY PRICING SCHEDULE		CALCULATE QUANTITY DISCOUNTS		PRICING GROUP	
		Pricing Schedule 1		By Line Quantity			
<input checked="" type="checkbox"/> USE MARGINAL RATES							
PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY	
Default Discount %			0.0%	5.0%	10.0%		
Base Price		50.00	50.00	52.50	55.00		
Alpha price level		75.00	75.00	78.75	82.50		
Alternate Price 1							
Alternate Price 2							
Alternate Price 3							
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50		
Beta price Level	50.0%	75.00	75.00	78.75	82.50		
Wholesale Price							
Online Price							

Use non-matrix Sublist APIs to get or set values for these pricing sublist fields.

**Figure 7. Figure 3 - Body Fields**

The screenshot shows the 'Pricing' sublist in a Dynamics 365 application. At the top, there are three dropdown menus: 'QUANTITY PRICING SCHEDULE' (set to 'Pricing Schedule 1'), 'CALCULATE QUANTITY DISCOUNTS' (set to 'By Line Quantity'), and 'PRICING GROUP'. A red box highlights the 'Pricing Group' dropdown and the 'USE MARGINAL RATES' checkbox below it. Below these are tabs for 'USA', 'British pound', 'Canadian Dollar', 'Euro', and 'Yen'. The main area is a grid where rows represent price levels and columns represent quantity levels (QTY 0, QTY 100, QTY 200, QTY 300). The grid contains numerical values representing prices.

Use standard Field APIs to get or set values for Pricing sublist body fields.

## Pricing Sublist Code Sample

This sample shows how to determine which pricing-related features are enabled in your account. It then shows how to programmatically determine the internal ID for the Pricing sublist itself, and then check to see if a Quantity Schedule has been applied to the items in this list. The script also shows how to set item prices and quantity levels depending on various conditions set within the pricing matrix.

**Note:** If your browser is inserting scroll bars in this code sample, maximize your browser window, or expand the main frame that this sample appears in.

```
// Check the features enabled in the account. See
Pricing Sublist Feature Dependencies for
// details on why this is important.
var multiCurrency = nlapiGetContext().getFeature('MULTICURRENCY');
var multiPrice = nlapiGetContext().getFeature('MULTPRICE');
var quantityPricing = nlapiGetContext().getFeature('QUANTITYPRICING');

// Set the name of the Price sublist based on features enabled and currency type.
// See
Pricing Sublist Internal IDs for details on why this is important.
var priceID;
var currencyID = "EUR";

// Set the ID for the sublist and the price field. Note that if all pricing-related features
// are disabled, you will set the price in the rate field. See
Pricing Sublist Feature Dependencies
// for details.

if (multiCurrency == 'F' && multiPrice == 'F' && quantityPricing == 'F')
    priceID = "rate";
else
{
    priceID = "price";
    if (multiCurrency == "T")
```

```

{
    var internalId = nlapiSearchRecord('currency', null, new nlobjSearchFilter('symbol',
        null,'contains', currencyID))[0].getId();

    // Append the currency ID to the sublist name
    priceID = priceID + internalId;
}
}

// Check to see if the item is using a Quantity Schedule
// If a Quantity Schedule is used, only the base price needs to be set.
// All other prices will be set according to the schedule.
var itemRecord = nlapiLoadRecord("inventoryitem", itemID);
var qtyPriceSchedule = itemRecord.getFieldValue('quantitypricingschedule');

// Set the base price
var basePrice = 100;

// You must select, set, and then commit the sublist line you want to change.
itemRecord.selectLineItem(priceID, 1);
itemRecord.setCurrentLineItemMatrixValue(priceID, 'price', 1, basePrice);
itemRecord.commitLineItem(priceID);

// Get the number of columns in the price matrix
// Each column represents a different quantity level
columnCount = itemRecord.getMatrixCount(priceID, 'price');

// Set the base price in each quantity of the price matrix for a specific sublist, e.g. currency
y

// Set the base price in each quantity
for (var j=1; j<=columnCount; j++)
{
    // Set the price for this cell of the matrix
    itemRecord.selectLineItem(priceID, 1);
    itemRecord.setCurrentLineItemMatrixValue(priceID, 'price', j, currencyBasePrice);
    itemRecord.commitLineItem( priceID );
}

// Display the full price matrix for a specific currency as an HTML table

// get the size of the matrix
var quantityLevels = itemRecord.getMatrixCount(priceID, 'price');
var priceLevels = itemRecord.getLineItemCount(priceID);
var priceName = "";
var priceNameField = "pricename";
var itemPrice = 0;
var fieldObj = null;

// create a table to present the results
var strName = "<table>";

if ( quantityLevels > 1 )

```

```

{
  strName += "<tr>";

  // write out the quantity levels as the first row
  for ( var j=1; j<=quantityLevels; j++)
  {
    strName += "<td>";

    // this Matrix API obtains an nlobjField object
    // the nlobjField object can be used to obtain the UI label
    fieldObj = itemRecord.getMatrixField( priceID, 'price', j);
    if (fieldObj != null )
      strName += fieldObj.getLabel();
    strName += j;
    strName += "</td>";

    strName += "<td>";

    // this Matrix API obtains the value of the Quantity level
    strName += itemRecord.getMatrixValue( priceID, 'price', j);
    strName += "</td>";
  }

  strName += "</tr>";
}

// loop through the matrix one row at a time
for ( var i=1; i<=priceLevels; i++)
{
  strName += "<tr>";

  // loop through each column of the matrix
  for ( j=1; j<=quantityLevels; j++)
  {
    // get the price for this cell of the matrix
    itemPrice = itemRecord.getLineItemMatrixValue( priceID, 'price', i, j);

    // Get the name of the price level. Note: you will use a standard
    // sublist API and not a matrix API for this.
    priceName = itemRecord.getLineItemText( priceID, priceNameField, i);

    strName += "<td>";
    strName += priceName;
    strName += "</td>";
    strName += "<td>";
    strName += itemPrice;
    strName += "</td>";
  }
  strName += "</tr>";
}

strName += "</table>";

```

# Matrix Sublist APIs and Standard Sublist APIs

When writing SuiteScript against the Pricing sublist, you may end up using different types of [Sublist APIs](#). If you want to get|set values in the pricing matrix, you will use [Matrix APIs](#).

If you want to get|set non-matrix fields, you will use all non-matrix [Sublist APIs](#) or [Field APIs](#), depending on which fields you are trying to access.

**Note:** See [What is the Pricing Matrix?](#) for information on the pricing matrix. See [Pricing Sublist Field IDs](#) for information on the differences among matrix, non-matrix, and body sublist fields.

## Matrix APIs

The following are considered to be **matrix APIs** for use on the Pricing sublist. Use these APIs to get|set matrix fields. See [Pricing Sublist Field IDs](#) to learn which fields are considered matrix fields.

Click these links to see the API documentation for each matrix API. Also see the figures below for a visual representation for where on the pricing matrix each matrix API executes.

- [nlapiGetMatrixField\(type, fldnam, column\)](#)
- [nlapiGetMatrixValue\(type, fldnam, column\)](#)
- [nlapiSetMatrixValue\(type, fldnam, column, value, firefieldchanged, synchronous\)](#)
- [nlapiGetMatrixCount\(type, fldnam\)](#)
- [nlapiGetCurrentLineItemMatrixValue\(type, fldnam, column\)](#)
- [nlapiSetCurrentLineItemMatrixValue\(type, fldnam, column, value, firefieldchanged, synchronous\)](#)
- [nlapiGetLineItemMatrixValue\(type, fldnam, linenum, column\)](#)
- [nlapiGetLineItemMatrixField\(type, fldnam, linenum, column\)](#)
- [nlapiFindLineItemMatrixValue\(type, fldnam, val, column\)](#)

PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300	QTY
Default Discount %			0.0%	5.0%	10.0%	
Base Price		50.00	50.00	52.50	55.00	
Alpha price level		75.00	75.00	78.75	82.50	
Alternate Price 1						
Alternate Price 2						
Alternate Price 3						
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50	
Beta price Level	50.0%	75.00	75.00	78.75	82.50	
Wholesale Price						
Online Price						

On matrix header fields, use [nlapiGetMatrixField](#), [nlapiGetMatrixValue](#), and [nlapiGetMatrixCount](#).

When on an existing line in the matrix, use [nlapiGetCurrentLineItemMatrixValue](#) and [nlapiSetCurrentLineItemMatrixValue](#) to get and set the price on that line in the specific column, respectively.

Pricing		CALCULATE QUANTITY DISCOUNTS		PRICING GROUP	
QUANTITY PRICING SCHEDULE	Pricing Schedule 1	By Line Quantity			
<input checked="" type="checkbox"/> USE MARGINAL RATES					
USA •	British pound •	Canadian Dollar •	Euro •	Yen •	
PRICE LEVEL	DEFAULT DISCOUNT %	QTY 0	QTY 100	QTY 200	QTY 300
Default Discount %			0.0%	5.0%	10.0%
Base Price		50.00	50.00	52.50	55.00
Alpha price level		75.00	75.00	78.75	82.50
Alternate Price 1					
Alternate Price 2					
Alternate Price 3					
Alternate Price 4	10.0%	55.00	55.00	57.75	60.50
Beta price Level	50.0%	75.00	75.00	78.75	82.50
Wholesale Price					
Online Price					

For all other lines in the matrix, use `nlapiGetLineItemMatrixValue`, `nlapiGetItemMatrixField`, and `nlapiFindLineItemValue`.

## Standard Sublist APIs

If you want to reference the other fields in the Pricing sublist, such as `currency`, `name`, or `discount`, use the existing `nlapiGetLineItemValue(...)` or `nlobjRecord.getLineItemValue(...)` APIs and pass in the existing `fldnam` (example: `price1currency`). Also see [Pricing Sublist Field IDs](#), which specifies which fields on the Pricing sublist can be set using standard [Sublist APIs](#).

**Example:**

```
// load an item record
var record = nlapiLoadRecord('inventoryitem', 536);

// get the value of the currency field on line 2
var currency = record.getLineItemValue('price1', 'currency', '2');

// get the value of the pricelevelname field on line 2
var pricelevelname2 = record.getLineItemValue('price1', 'pricelevel', '2');
var pricelevelname3 = record.getLineItemValue('price1', 'pricelevel', '3');
var pricelevelname4 = record.getLineItemValue('price1', 'pricelevel', '4');

// returns the discount from line item 2
var discount2 = record.getLineItemValue('price1', 'discount', '2');
var discount3 = record.getLineItemValue('price1', 'discount', '3');
var discount4 = record.getLineItemValue('price1', 'discount', '4');
```

## Records that Include the Pricing Sublist

The Pricing sublist appears on the following records: Assembly Item, Lot Numbered Assembly Item, Serialized Assembly Item, Lot Numbered Inventory Item, Service Sale Item, Other Charge Sale Item, Serialized Inventory Item, Gift Certificate Item, Kit Item, Inventory Item, Non Inventory Sale Item, Non Inventory Resale Item, Other Charge Resale Item, Service Resale Item.

## Predecessors Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **predecessor**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Predecessors sublist appears on the project task record. To see the internal IDs associated with the Predecessors sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Related Solutions Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **solutions**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Related Solutions sublist appears on the solution record. To see the internal IDs associated with the Related Solutions sublist, refer to the [SuiteScript Records Browser](#).

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Resources Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **resource**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Resources sublist appears on the event record. To see the internal IDs associated with the Resources sublist, open the [SuiteScript Records Browser](#) and click on the Event record.

**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Sales Team Sublist

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **salesteam**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)



The Sales Team sublist appears on the following records: Cash Refund, Cash Sale, Customer, Credit Memo, Estimate/Quote, Invoice, Opportunity, Return Authorization, Sales Order, Work Order. To see the internal IDs associated with the Sales Team sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Shipping Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **shipgroup**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Shipping sublist appears on the following records: Sales Order, Cash Sale, Invoice, Estimate, and Item Fulfillment. (Note that although the Shipping sublist is supported on the Item Fulfillment record type, this sublist is not currently showing on this record in the [SuiteScript Records Browser](#). To get the internal IDs for the Shipping sublist, open the Records Browser and navigate to one of the other record types that support this sublist.)



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Usage Notes

Sublist Field Internal ID	Sublist Field UI Label	Note
shippingtaxcode	Shipping Tax Code	This sublist field appears only if per-line taxes have been set on the Item sublist.

## Site Category



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **siterecategory**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Site Category sublist appears on the following records: Assembly Item, Download Item, Gift Certificate Item, Inventory Part, Kit Item, Lot Numbered Assembly Item, Lot Numbered Inventory Item, Serialized Assembly Item, Serialized Inventory Item, Service Item. To see the internal IDs associated with the Site Category sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Time Tracking Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **timeitem**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Time Tracking sublist appears on the following records: Event, Customer, Project (Job), Lead, Phone Call, Prospect, Support Case, Task.

Note that although the Time Tracking sublist is supported on the Customer, Project, Lead, and Prospect record types, this sublist is not currently showing on these records in the SuiteScript Records Browser. To get the internal IDs for the Time Tracking sublist, open the [Records Browser](#) and navigate to any of the other record types that support this sublist.) For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.



**Important:** If any case, task, or event record has more than 9500 time entries in the Time Tracking sublist, all cases include a static list of time entries. These static lists are not accessible to scripting. Attempts to script on Time Tracking sublists that are static lists of time entries will not return accurate results.

## Topics Sublist



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **topics**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Topics sublist appears on the Solution record. To see the internal IDs associated with the Topics sublist, refer to the [SuiteScript Records Browser](#).



**Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Units



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is **uom**. This sublist is an **inline editor** sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Units sublist appears on the Unit of Measure (Unit Type) record. To see the internal IDs associated with the Units sublist, refer to the [SuiteScript Records Browser](#).

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Vendors

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The internal ID for this sublist is `itemvendor`. This sublist is an [inline editor](#) sublist. (In the NetSuite Help Center, see [Inline Editor Sublists](#) for information on this sublist type.)

The Vendors sublist appears on the following records: Lot Numbered Assembly Item and Serialized Assembly Item. To see the internal IDs associated with the Vendors sublist, open the [SuiteScript Records Browser](#) and navigate to one of the records that includes this sublist.

**i Note:** For information on using the SuiteScript Records Browser, see [Working with the SuiteScript Records Browser](#) in the NetSuite Help Center.

## Related Items Sublist

**i Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

Field UI Label	Field Internal ID	Field Type	Mandatory	Field Notes
Base Price	baseprice	currency	false	
Item Description	description	textarea	false	
Item	item	select	false	
Online Price	onlineprice	currency	false	
Related Item	presentationitem	select	true	

# Record Initialization Defaults

You can use SuiteScript to specify record initialization parameters that will default when creating, copying, loading, and transforming records. To enable this behavior, use the `initializeValues` parameter in the following APIs:

- `nlapiCreateRecord(type, initializeValues)`
- `nlapiCopyRecord(type, id, initializeValues)`
- `nlapiDeleteRecord(type, id, initializeValues)`
- `nlapiLoadRecord(type, id, initializeValues)`

In `nlapiTransformRecord(type, id, transformType, transformValues)`, use the `transformValues` parameter to set initialization values during the record transformation process.

The `initializeValues` parameter is an Object that can contain an array of name/value pairs of defaults that are passed upon record initialization. The following table lists initialization types that are available to certain SuiteScript-supported records and the values they can contain. For examples, see [Record Initialization Examples](#).

 **Important:** In your scripts, the property type does not need to be in quotes, but the property value does, unless it is a variable, number, or boolean.

Record	Initialization Type	Values
All SuiteScript-supported records. For a list of records, see <a href="#">SuiteScript Supported Records</a> . For information on scripting a record in dynamic mode, see the help topic <a href="#">Working with Records in Dynamic Mode</a> .	recordmode	dynamic
All SuiteScript-supported records that support form customization.	customform	<customformid>
All transactions	deletionreason deletionreasonmemo	<deletionreasonid> <string>
 <b>Important:</b> Use of deletionreasoncode and deletionreasonmemo is supported only for <code>nlapiDeleteRecord</code> . It is not supported for <code>nlapiCopyRecord</code> , <code>nlapiCreateRecord</code> , or <code>nlapiLoadRecord</code> .		
Assembly Build	assemblyitem	<assemblyitemid>
Cash Refund	entity	<entityid>
Cash Sale	entity	<entityid>
Check	entity	<entityid>
Credit Memo	entity	<entityid>
Customer Payment	entity	<entityid>   <inv>
Customer Refund	entity	<entityid>
Estimate	entity	<entityid>

Record	Initialization Type	Values
Expense Report	entity	<entityid>
Invoice	entity	<entityid>
Item Receipt	entity	<entityid>
Non-Inventory Part	subtype	sale   resale   purchase
Opportunity	entity	<entityid>
Other Charge Item	subtype	sale   resale   purchase
Purchase Order	entity	<entityid>
Return Authorization	entity	<entityid>
Sales Order	entity	<entityid>
Script Deployment	script	<scriptid>
Service	subtype	sale   resale   purchase
Tax Group	nexuscountry	<countrycode> See <a href="#">Country Codes</a> Used for Initialization Parameters.
Tax Type	country	<countrycode> See <a href="#">Country Codes</a> Used for Initialization Parameters.
Topic	parenttopic	<parenttopicid>
Vendor Bill	entity	<entityid>
Vendor Payment	entity	<entityid>
Work Order	assemblyitem	<assemblyitemid>

## Record Initialization Examples

The following samples show multiple ways to specify record initialization values. You can specify one name/value pair at a time or an array of name/value pairs.

### Example 1

```
//load a sales order in dynamic mode
var rec = nlapiLoadRecord('salesorder', {recordmode: 'dynamic'});
```

### Example 2

```
//load a sales order in dynamic mode and source in all values from customer (entity) 87
var rec = nlapiLoadRecord('salesorder', 111, {recordmode: 'dynamic', entity: 87});
```

### Example 2

```
//create a sales order that uses values from custom form 17
```



```
var rec = nlapiCreateRecord('salesorder', {customform: 17});
```

### Example 3

```
//copy a sales order – the record object returned from the copy will be in dynamic  
//mode and contain values from custom form 17  
var rec = nlapiCopyRecord('salesorder', 55, {recordmode: 'dynamic', customform: 17});
```

### Example 4

```
//create an Array to set multiple initialization values  
var initvalues = new Array();  
initvalues.customform= 17;  
initvalues.recordmode = 'dynamic';  
initvalues.entity = 355;  
  
// create sales order and pass values stored in the initvalues array  
var rec = nlapiCreateRecord('salesorder', initvalues);
```

### Example 5

```
//create a script deployment record in dynamic mode and attach a script record to it  
var rec = nlapiCreateRecord('scriptdeployment', {recordmode: 'dynamic', script: 68});  
rec.setFieldValue('status', 'NOTSCHEDULED');
```

## Country Codes Used for Initialization Parameters

If you are scripting the Tax Group or Tax Type records, you can initialize the record to source all values related to a specific country. In your script, use the country code for the <countrycodeid> value, for example:

```
nlapiCreateRecord('taxgroup', {nexuscountry: 'AR'});
```

Country Code	Country Name
AD	Andorra
AE	United Arab Emirates
AF	Afghanistan
AG	Antigua and Barbuda
AI	Anguilla
AL	Albania
AM	Armenia
AN	Netherlands Antilles
AO	Angola
AQ	Antarctica
AR	Argentina
AS	American Samoa

Country Code	Country Name
AT	Austria
AU	Australia
AW	Aruba
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BD	Bangladesh
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BH	Bahrain
BI	Burundi
BJ	Benin
BL	Saint Barthélemy
BM	Bermuda
BN	Brunei Darussalam
BO	Bolivia
BR	Brazil
BS	Bahamas
BT	Bhutan
BV	Bouvet Island
BW	Botswana
BY	Belarus
BZ	Belize
CA	Canada
CC	Cocos (Keeling) Islands
CD	Congo, Democratic People's Republic
CF	Central African Republic
CG	Congo, Republic of
CH	Switzerland
CI	Cote d'Ivoire
CK	Cook Islands
CL	Chile
CM	Cameroon
CN	China

Country Code	Country Name
CO	Colombia
CR	Costa Rica
CU	Cuba
CV	Cap Verde
CX	Christmas Island
CY	Cyprus
CZ	Czech Republic
DE	Germany
DJ	Djibouti
DK	Denmark
DM	Dominica
DO	Dominican Republic
DZ	Algeria
EC	Ecuador
EE	Estonia
EG	Egypt
EH	Western Sahara
ER	Eritrea
ES	Spain
ET	Ethiopia
FI	Finland
FJ	Fiji
FK	Falkland Islands (Malvina)
FM	Micronesia, Federal State of
FO	Faroe Islands
FR	France
GA	Gabon
GB	United Kingdom (GB)
GD	Grenada
GE	Georgia
GF	French Guiana
GG	Guernsey
GH	Ghana
GI	Gibraltar
GL	Greenland



Country Code	Country Name
GM	Gambia
GN	Guinea
GP	Guadeloupe
GQ	Equatorial Guinea
GR	Greece
GS	South Georgia
GT	Guatemala
GU	Guam
GW	Guinea-Bissau
GY	Guyana
HK	Hong Kong
HM	Heard and McDonald Islands
HN	Honduras
HR	Croatia/Hrvatska
HT	Haiti
HU	Hungary
ID	Indonesia
IE	Ireland
IL	Israel
IM	Isle of Man
IN	India
IO	British Indian Ocean Territory
IQ	Iraq
IR	Iran (Islamic Republic of)
IS	Iceland
IT	Italy
JE	Jersey
JM	Jamaica
JO	Jordan
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KH	Cambodia
KI	Kiribati
KM	Comoros



Country Code	Country Name
KN	Saint Kitts and Nevis
KP	Korea, Democratic People's Republic
KR	Korea, Republic of
KW	Kuwait
KY	Cayman Islands
KZ	Kazakhstan
LA	Lao People's Democratic Republic
LB	Lebanon
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
LY	Libyan Arab Jamahiriya
MA	Morocco
MC	Monaco
MD	Moldova, Republic of
ME	Montenegro
MF	Saint Martin
MG	Madagascar
MH	Marshall Islands
MK	Macedonia
ML	Mali
MM	Myanmar
MN	Mongolia
MO	Macau
MP	Northern Mariana Islands
MQ	Martinique
MR	Mauritania
MS	Montserrat
MT	Malta
MU	Mauritius



Country Code	Country Name
MV	Maldives
MW	Malawi
MX	Mexico
MY	Malaysia
MZ	Mozambique
NA	Namibia
NC	New Caledonia
NE	Niger
NF	Norfolk Island
NG	Nigeria
NI	Nicaragua
NL	Netherlands
NO	Norway
NP	Nepal
NR	Nauru
NU	Niue
NZ	New Zealand
OM	Oman
PA	Panama
PE	Peru
PF	French Polynesia
PG	Papua New Guinea
PH	Philippines
PK	Pakistan
PL	Poland
PM	St. Pierre and Miquelon
PN	Pitcairn Island
PR	Puerto Rico
PS	Palestinian Territories
PT	Portugal
PW	Palau
PY	Paraguay
QA	Qatar
RE	Reunion Island
RO	Romania



Country Code	Country Name
RS	Serbia
RU	Russian Federation
RW	Rwanda
SA	Saudi Arabia
SB	Solomon Islands
SC	Seychelles
SD	Sudan
SE	Sweden
SG	Singapore
SH	St. Helena
SI	Slovenia
SJ	Svalbard and Jan Mayen Islands
SK	Slovak Republic
SL	Sierra Leone
SM	San Marino
SN	Senegal
SO	Somalia
SR	Suriname
ST	Sao Tome and Principe
SV	El Salvador
SY	Syrian Arab Republic
SZ	Swaziland
TC	Turks and Caicos Islands
TD	Chad
TF	French Southern Territories
TG	Togo
TH	Thailand
TJ	Tajikistan
TK	Tokelau
TM	Turkmenistan
TN	Tunisia
TO	Tonga
TP	East Timor
TR	Turkey
TT	Trinidad and Tobago

Country Code	Country Name
TV	Tuvalu
TW	Taiwan
TZ	Tanzania
UA	Ukraine
UG	Uganda
UM	US Minor Outlying Islands
US	United States
UY	Uruguay
UZ	Uzbekistan
VA	Holy See (City Vatican State)
VC	Saint Vincent and the Grenadines
VE	Venezuela
VG	Virgin Islands (British)
VI	Virgin Islands (USA)
VN	Vietnam
VU	Vanuatu
WF	Wallis and Futuna Islands
WS	Western Samoa
YE	Yemen
YT	Mayotte
ZA	South Africa
ZM	Zambia
ZW	Zimbabwe

# Permission Names and IDs



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table provides permission names and IDs associated with each NetSuite feature. You can use the permission ID with nlobjContext.getSetting(type, name) to return the permission levels that have been specified in your account.

Permission ID	Permission Name	Feature
ADMI_ACCOUNTING	Set Up Accounting	Accounting
ADMI_ACCOUNTINGBOOK	Accounting Book	Accounting
ADMI_ACCOUNTINGLIST	Accounting Lists	Accounting
ADMI_ACCTPERIODS	Manage Accounting Periods	Accounting Periods
ADMI_ACCTSETUP	Set Up Accounting	Accounting
ADMI_ACH	Set Up ACH Processing	ACH Processing
ADMI_ADVANCED_ORDER_MANAGEMENT	Advanced Order Management	Advanced Order Management
ADMI_APPPUBLISHER	Application Publishers	SSP Applications
ADMI_AUDITLOGIN	View Login Audit Trail	Login Audit Trail
ADMI_BACKUPEXPORT	Backup Your Data	CSV Export
ADMI_BILLPAYSETUP	Set Up Bill Pay	Online Bill Pay
ADMI_BUNDLER	SuiteBundler	SuiteBundler
ADMI_BUNDLERMANUP	SuiteBundler Upgrade Install Base	SuiteBundler
ADMI_CAMPAIGNEMAIL	Set Up Campaign Email Addresses	Marketing Automation
ADMI_CAMPAIGNSETUP	Setup Campaigns	Marketing Automation
ADMI_CASEFORM	Online Case Form	Customer Support and Service
ADMI_CASEISSUE	Support Case Issue	Customer Support and Service
ADMI_CASEORIGIN	Support Case Origin	Customer Support and Service
ADMI_CASEPRIORITY	Support Case Priority	Customer Support and Service
ADMI_CASERULE	Support Case Territory Rule	Customer Support and Service
ADMI_CASESTATUS	Support Case Status	Customer Support and Service
ADMI_CASETERRITORY	Support Case Territory	Customer Support and Service
ADMI_CASETYPE	Support Case Type	Customer Support and Service
ADMI_CLASSESTOLOCs	Convert Classes to Locations	Locations
ADMI_CLASSSEGMENTMAPPING	Class Segment Mapping	Classes
ADMI_CLOSEPERIOD	Lock Transactions	Accounting Periods
ADMI_COMMISSIONSETUP	Commission Feature Setup	Commissions

Permission ID	Permission Name	Feature
ADMI_COMPANY	Company Information	Company Setup
ADMI_CONVERTCLASSES	Convert Classes to Departments	Departments
ADMI_CONVERTLEAD	Lead Conversion Mapping	Sales Force Automation
ADMI_COPYPROJECTTASK	Copy Project Tasks	Project Management
ADMI_CREATEPEER	Copy Chart of Accounts to New Company	Accounting
ADMI_CREDITCARD	Credit Card Processing	Credit Card Payments
ADMI_CRMLIST	CRM Lists	Opportunities
ADMI_CSIMPORTPREF	Set Up CSV Preferences	Accounting
ADMI_CUSTADDRESSFORM	Custom Address Form	Custom Forms
ADMI_CUSTBODYFIELD	Custom Body Fields	Custom Fields
ADMI_CUSTCATEGORY	Custom Center Categories	Custom Centers
ADMI_CUSTCENTER	Custom Centers	Custom Centers
ADMI_CUSTCOLUMNFIELD	Custom Column Fields	Custom Fields
ADMI_CUSTEMAILLAYOUT	Custom HTML Layouts	Custom PDF/HTML Templates
ADMI_CUSTENTITYFIELD	Custom Entity Fields	Custom Fields
ADMI_CUSTENTRYFORM	Custom Entry Forms	Custom Forms
ADMI_CUSTEVENTFIELD	Custom Event Fields	Custom Fields
ADMI_CUSTFIELD	Custom Fields	SuiteFlow Custom Fields
ADMI_CUSTFIELDTAB	Custom Subtabs	Custom Subtabs
ADMI_CUSTFORM	Custom Transaction Forms	Custom Forms
ADMI_CUSTITEMFIELD	Custom Item Fields	Custom Fields
ADMI_CUSTITEMNUMBERFIELD	Custom Item Number Fields	Inventory
ADMI_CUSTLAYOUT	Custom PDF Layouts	Custom PDF/HTML Templates
ADMI_CUSTLIST	Custom Lists	Custom Lists
ADMI_CUSTOMERFORM	Online Customer Form	Sales Force Automation
ADMI_CUSTOMSCRIPT	SuiteScript	Client SuiteScript
ADMI_CUSTOMSUBLIST	Custom Sublists	Custom Sublists
ADMI_CUSTOTHERFIELD	Other Custom Fields	Custom Fields
ADMI_CUSTRECORD	Custom Record Types	Custom Records
ADMI_CUSTRECORDFORM	Online Custom Record Form	Custom Forms
ADMI_CUSTSECTION	Custom Center Tabs	Custom Records
ADMI_CUSTTASKS	Custom Center Links	Custom Records
ADMI_CUSTTRANSACTION	Custom Transaction Types	Custom Transactions
ADMI_DELETEDRECORD	Deleted Records	Search for Deleted Records

Permission ID	Permission Name	Feature
ADMI_DEPTSEGMENTMAPPING	Department Segment Mapping	Departments
ADMI_DEVICE_ID	Device ID Management	Suite Commerce InStore
ADMI_DOMAINS	Set Up Domains	SuiteCommerce
ADMI_DUPLICATESETUP	Duplicate Detection Setup	Duplicate Detection & Merge
ADMI_EMPLCATEGORY	Publish Employee List	SuiteCommerce
ADMI_EMPLOYEELIST	Other Lists	Accounting
ADMI_ENABLEFEATURES	Enable Features	Enable Company Features
ADMI_ENTITYACCOUNTMAPPING	Entity Account Mapping	Accounting
ADMI_ENTITYSTATUS	Customer Status	Customer Relationship Management
ADMI_ESCALATIONRULE	Escalation Assignment Rule	Customer Support and Service
ADMI_ESCALATIONTERRITORY	Escalation Assignment	Customer Support and Service
ADMI_EXPORTIIF	Export as IIF	Accounting
ADMI_EXPORTIMPORTEBAY	eBay Export/Import	eBay Integration
ADMI_FINCHARGEPRF	Finance Charge Preferences	A/R
ADMI_GLOBALACCOUNTMAPPING	Global Account Mapping	Accounting
ADMI_IMPORTCSVFILE	Import CSV File	CSV Import
ADMI_IMPORTDEFAULT	Delete All Data	Delete All Data
ADMI_IMPORTQIFFILE	Import Quicken QIF File	Accounting
ADMI_IMPORTXML	Set Up ADP Payroll	Import ADP Payroll
ADMI_INTEGRAPP	Integration Application	Integration
ADMI_ISSUESETUP	Issue Setup	Issue Management
ADMI_ITEMACCOUNTMAPPING	Item Account Mapping	Accounting
ADMI_KNOWLEDGEBASE	Publish Knowledge Base	Knowledge Base
ADMI_KPIREPORT	KPI Scorecards	KPI Scorecards
ADMI_LOCATIONCOSTINGGROUP	Location Costing Group	
ADMI_LOCSEGMENTMAPPING	Location Segment Mapping	Custom Segments
ADMI_MANAGE_OAUTH_TOKENS	Access Token Management	Token-based Authentication
ADMI_MANAGE_OWN_OAUTH_TOKENS	User Access Tokens	Token-based Authentication
ADMI_MANAGECUSTOMSEGMENTS	Custom Segments	Custom Segments
ADMI_MANAGEROLES	Bulk Manage Roles	Show Role Differences
ADMI_MANAGEUSERS	Manage Users	Managing Users

Permission ID	Permission Name	Feature
ADMI_OPENIDSSOSETUP	Set Up OpenID Single Sign-on	OpenID Single Sign-on
ADMI_OUTLOOKINTEGRATION	Outlook Integration	Outlook Integration
ADMI_OUTLOOKINTEGRATION_V3	Outlook Integration 3.0	Outlook Integration
ADMI_PAYROLL	Set Up Payroll	Payroll
ADMI_PUBLISHER_APP	Publisher Applications	SuiteCommerce
ADMI_PUBLISHER_ID	Publisher ID	SuiteCommerce
ADMI_SALESTERRITORY	Sales Territory	Sales Force Automation
ADMI_SAMLSSOSETUP	Set Up SAML Single Sign-on	SAML Single Sign-on
ADMI_SAVEDASHBOARD	Publish Dashboards	Publishing Dashboards
ADMI_SETUPCOMPANY	Set Up Company	Company Preferences
ADMI_SETUPEBAY	Set Up eBay	eBay Integration
ADMI_SETUPIMAGERESIZE	Set Up Image Resizing	SuiteCommerce Advanced
ADMI_SETUPYEARSTATUS	Set Up Year Status	Accounting
ADMI_SFASETUP	Sales Force Automation Setup	Sales Force Automation
ADMI_STATETAXIMPORT	Import State Sales Tax	Accounting
ADMI_STORESETUP	Set Up Web Site	Web Site
ADMI_SUITEANALYTICSCONNECT	SuiteAnalytics Connect	SuiteAnalytics Connect
ADMI_SUITESIGNON	SuiteSignOn	Outbound Single Sign-on
ADMI_SUPPORTSETUP	Support Setup	Customer Support and Service
ADMI_SWAPPICES	Swap Prices Between Price Levels	Multiple Prices
ADMI_SYNCHRONIZATIONSETUP	Set up synchronization	Synchronization
ADMI_TAXPERIODS	Manage Tax Reporting Periods	Accounting
ADMI_TRANSITEMTXT	Translation	Multi-Language
ADMI_TWOFACTORAUTH	Two-Factor Authentication	Two-Factor Authentication
ADMI_TWOFACTORAUTHBASE	Two-Factor Authentication base	Two-Factor Authentication
ADMI_UNCATSITEITEMS	Uncategorized Presentation Items	Web Store
ADMI_UPDATEPRICES	Update Prices	Item Record Management
ADMI_UPSELLSETUP	Upsell Setup	Upsell Manager
ADMI_USERPREF	User Preferences	Set Preferences
ADMI_WEBSERVICES	Web Services	SuiteTalk
ADMI_WEBSERVICESSETUP	Set Up Web Services	SuiteTalk
ADMI_WORKFLOW	Workflow	SuiteFlow

Permission ID	Permission Name	Feature
LIST_ACCOUNT	Accounts	Accounting
LIST_AMORTIZATION	Amortization Schedules	Amortization
LIST_BIG_SEARCH	Persist Search	Search
LIST_BILLINGSCHEDULE	Billing Schedules	Advanced Billing
LIST_BILLOFDISTRIBUTION	Bill Of Distribution	Advanced Inventory Management
LIST_BILLOFMATERIALSINQUIRY	Bill Of Materials Inquiry	Assembly Items
LIST_BIN	Bins	Accounting
LIST_CALENDAR	Calendar	Calendar Preferences
LIST_CALL	Phone Calls	Phone Calls
LIST_CAMPAIGN	Marketing Campaigns	Marketing Automation
LIST_CAMPAIGNHISTORY	Campaign History	Marketing Automation
LIST_CASE	Cases	Customer Support and Service
LIST_CATEGORY	Expense Categories	Expense Reports
LIST_CHECKITEMAVAILABILITY	Check Item Availability	Advanced Inventory Management
LIST_CLASS	Classes	Classes
LIST_COLORTHEME	Color Themes	SuiteCommerce
LIST_COMMISSIONRULES	Employee Commission Schedules/Plans	Employee Commissions
LIST_COMPANY	Companies	Customer Relationship Management
LIST_COMPETITOR	Competitors	Sales Force Automation
LIST_COMPONENTWHEREUSEDINQUIRY	Component Where Used	Assembly Items
LIST_CONTACT	Contacts	Contacts
LIST_CONTACTROLE	Contact Roles	Sales Force Automation
LIST_COSTEDBOMINQUIRY	Costed Bill Of Materials Inquiry	Assembly Items
LIST_CRMGROUP	CRM Groups	Customer Relationship Management
LIST_CRMMESSAGE	Track Messages	Customer Relationship Management
LIST_CRMTEMPLATE	Marketing Template	Marketing Automation
LIST_CURRENCY	Currency	Multiple Currencies
LIST_CUSTJOB	Customers	Prospects and Contacts
LIST_CUSTPROFILE	Customer Profile	Customer Profile
LIST_CUSTRECORDENTRY	Custom Record Entries	Custom Records
LIST_DEPARTMENT	Departments	Departments
LIST_DISTRIBUTIONNETWORK	Distribution Network	Advanced Inventory Management

Permission ID	Permission Name	Feature
LIST_EMAILTEMPLATE	Email Template	Email Template
LIST_EMPLOYEE	Employees	Employees
LIST_EMPLOYEESSN	Employee Social Security Numbers	Employees
LIST_ENTITY_DUPLICATES	Duplicate Entity Management	Duplicate Detection & Merge
LIST_EVENT	Events	Events
LIST_EXPORT	Export Lists	Search Result Export
LIST_FAIRVALUEDIMENSION	Fair Value Dimension	Revenue Recognition
LIST_FAIRVALUEFORMULA	Fair Value Formula	Revenue Recognition
LIST_FAIRVALUEPRICE	Fair Value Price	Revenue Recognition
LIST_FAXMESSAGE	Fax Messages	Communications
LIST_FAXTEMPLATE	Fax Template	Mail Merge
LIST_FILECABINET	Documents and Files	File Cabinet
LIST_FIND	Perform Search	Order Management
LIST_FISCALCALENDAR	Fiscal Calendars	Accounting
LIST_GENERICRESOURCE	Generic Resources	Project Management
LIST_GLLINESAUDITLOG	Custom GL Lines Plug-in Audit Log	Custom GL Lines Plug-in
LIST_GLLINESAUDITLOGSEG	Custom GL Lines Plug-in Audit Log (Segments)	Custom GL Lines Plug-in
LIST_HISTORY	Notes Tab	Communications
LIST_INFOITEM	Store Content Items	SuiteCommerce
LIST_INFOITEMFORM	Publish Forms	SuiteCommerce
LIST_INTEGRAPP	Integration Applications	SuiteTalk
LIST_INVCOSTTEMPLATE	Inventory Cost Template	Multi-Book Accounting
LIST_ISSUE	Issues	Issue Management
LIST_ITEM	Items	Item Record Management
LIST_ITEM_REVISION	Item Revisions	Item Record Management
LIST_ITEMDEMANDPLAN	Item Demand Plan	Advanced Inventory Management
LIST_ITEMREVENUECATEGORY	Item Revenue Category	Revenue Recognition
LIST_ITEMSUPPLYPLAN	Item Supply Plan	Advanced Inventory Management
LIST_JOB	Jobs	Projects
LIST_KNOWLEDGEBASE	Knowledge Base	Knowledge Base
LIST_LOCATION	Locations	Locations
LIST_MAILMERGE	Mail Merge	Mail Merge
LIST_MAILMESSAGE	Letter Messages	Communications

Permission ID	Permission Name	Feature
LIST_MAILTEMPLATE	Letter Template	Mail Merge
LIST_CRMTEMPLATE	Marketing Template	
LIST_MASSUPDATES	Mass Updates	Mass Update
LIST_MEDIAITEMFOLDER	Media Folders	File Cabinet
LIST_MEMDOC	Memorized Transactions	Transactions
LIST_MFGCOSTTEMPLATE	Manufacturing Cost Template	Inventory Management
LIST_MFGROUTING	Manufacturing Routing	Inventory Management
LIST_NOTIFICATION	Notifications	
LIST_ORDER_REALLOCATION	Commit Orders	Advanced Inventory Management
LIST_OTHERNAME	Other Names	Sales Force Automation
LIST_PARTNER	Partners	Partner Relationship Management
LIST_PARTNERCOMMISSNRULES	Partner Commission Schedules/Plans	Partner Commissions/Royalties
LIST_PAYCHECK	Paychecks	Payroll
LIST_PAYMETH	Payment Methods	Order Management
LIST_PAYROLLITEM	Payroll Items	Payroll
LIST_PDFMESSAGE	PDF Messages	Communications
LIST_PDFTEMPLATE	PDF Template	Mail Merge
LIST_PLANNEDREVENUE	Planned Revenue	Revenue Recognition
LIST_PLANNEDSTANDARDCOST	Planned Standard Cost	Item Record Management
LIST_PRESCATEGORY	Presentation Categories	SuiteCommerce
LIST_PROJECTREVENUERULE	Project Revenue Rules	
LIST_PROJECTTASK	Project Tasks	Project Management
LIST_PROJECTTEMPLATE	Project Templates	Project Management
LIST_PROMOTIONCODE	Promotion Code	Sales Force Automation
LIST_PUBLISHSEARCH	Publish Search	Publishing Search Results
LIST_QUANTITYPRICINGSCHEDULE	Quantity pricing Schedules	Quantity Pricing
LIST RELATEDITEMS	Related Items	Web Site
LIST_RESOURCE	Resource	Project Management
LIST_REVENUEELEMENT	Revenue Element	Revenue Recognition
LIST_REVENUEPLAN	Revenue Recognition Plan	Revenue Recognition
LIST_REVENUERECOGNITIONRULE	Revenue Recognition Rule	Revenue Recognition
LIST_REVRECSCHEDULE	Revenue Recognition Schedules	

Permission ID	Permission Name	Feature
LIST_REVRECFIELDMAPPING	Revenue Recognition Field Mapping	Revenue Recognition
LIST_REVRECSCHEDULE	Revenue Recognition Schedules	Revenue Recognition
LIST_REVRECVSOE	Revenue Management VSOE	VSOE
LIST_RSRCALLOCATION	Resource Allocations	Project Management
LIST_RSRCALLOCATIONAPPRV	Resource Allocation Approval	Project Management
LIST_SALESCAMPAIGN	Sales Campaigns	Sales Campaigns
LIST_SALESROLE	Sales Roles	Team Selling
LIST_SHIPITEM	Shipping Items	Accounting
LIST_SHIPPARTPACKAGE	Shipping Partner Package	Order Management
LIST_SHIPPARTREGISTRATION	Shipping Partner Registration	Order Management
LIST_SHIPPARTSHIPMENT	Shipping Partner Shipment	Order Management
LIST_SITEEMAILTEMPLATE	Web Store Email Template	SuiteCommerce
LIST_STANDARDCOSTVERSION	Standard Cost Version	Item Record Management
LIST_STORETAB	Store Tabs	Web Store
LIST_SUBSIDIARY	Subsidiaries	Subsidiaries
LIST_SYSTEMEMAILTEMPLATE	System Email Template	Customer Relationship Management
LIST_TASK	Tasks	Customer Relationship Management
LIST_TAXITEM	Tax Items	Accounting
LIST_TAXSCHEDULE	Tax Schedules	Advanced Taxes
LIST_TEGATAACCOUNT	Tegata Accounts	Accounting
LIST_TEMPLATE_CATEGORY	Template Categories	Email Marketing Campaigns
LIST_TRANNUMBERAUDITLOG	Access to transaction numbering audit log	Transactions
LIST_UNIT	Units	Accounting
LIST_UPSELL	Upsell Assistant	Upsell Manager
LIST_UPSELLWIZARD	Upsell Wizard	Upsell Manager
LIST_VENDOR	Vendors	Accounting
LIST_WEBSITE	Website (External) publisher	Web Site
LIST_WORKCALENDAR	Work Calendar	Project Management
LIST_WORKPLACE	Workplaces	Payroll
REGT_ACCTPAY	Accounts Payable Register	A/P
REGT_ACCTREC	Accounts Receivable Register	A/R
REGT_BANK	Bank Account Registers	Accounting
REGT_COGS	Cost of Goods Sold Registers	Accounting

Permission ID	Permission Name	Feature
REGT_CREDCARD	Credit Card Registers	Accounting
REGT_DEFEREXPENSE	Deferred Expense Registers	Amortization
REGT_DEFERREVENUE	Deferred Revenue Registers	Revenue Recognition
REGT_EQUITY	Equity Registers	Accounting
REGT_EXPENSE	Expense Registers	Accounting
REGT_FIXEDASSET	Fixed Asset Registers	Accounting
REGT_INCOME	Income Registers	Accounting
REGT_LONGTERMLIAB	Long Term Liability Registers	Accounting
REGT_NONPOSTING	Non Posting Registers	Accounting
REGT_OTHASSET	Other Asset Registers	Accounting
REGT_OTHCURRASSET	Other Current Asset Registers	Accounting
REGT_OTHCURRLIAB	Other Current Liability Registers	Accounting
REGT_OTHINCOME	Other Income Registers	Accounting
REGT_UNBILLEDREC	Unbilled Receivable Registers	Revenue Commitments
REPO_1099	Form 1099 - Federal Miscellaneous Income	A/P
REPO_940	Form 940 - Employer's Annual Federal Unemployment Tax Return	Payroll
REPO_941	Form 941 - Employer's Quarterly Federal Tax Return	Payroll
REPO_ACCOUNTDETAIL	Account Detail	Accounting
REPO_AMORTIZATION	Amortization Reports	Amortization
REPO_AP	Accounts Payable	A/P
REPO_AR	Accounts Receivable	A/R
REPO_AUTHPARTNERCOMMISSION	Partner Authorized Commission Reports	Partner Commissions/Royalties
REPO_BALANCESHEET	Balance Sheet	Accounting
REPO_BOOKINGS	Sales Order Reports	Sales Force Automation
REPO_CASHFLOW	Cash Flow Statement	Accounting
REPO_COMMISSION	Commission Reports	Employee Commissions
REPO_CUSTOMIZATION	Report Customization	Report Customization
REPO_FINANCIALS	Financial Statements	Accounting
REPO_GL	General Ledger	Accounting
REPO_INTEGRATION	Integration	SuiteTalk
REPO_INVENTORY	Inventory	Inventory
REPO_ISSUE	Issue Reports	Issue Management



Permission ID	Permission Name	Feature
REPO_MARKETING	Marketing Campaign Reports	Marketing Automation
REPO_NONPOSTING	Sales Order Transaction Report	Order Management
REPO_PANDL	Income Statement	Accounting
REPO_PARTNERCOMMISSION	Partner Commission Reports	Partner Commissions/Royalties
REPO_PARTNERMONITOR	Partner Monitor Results	Partners
REPO_PAYCHECKDETAIL	Payroll Check Register	Payroll
REPO_PAYROLL	Payroll Summary & Detail Reports	Payroll
REPO_PAYROLLHOURSEARNING	Payroll Hours & Earnings	Payroll
REPO_PAYROLLJOURNAL	Payroll Journal Report	Payroll
REPO_PAYROLLLIAB	Payroll Liability Report	Payroll
REPO_PAYROLLSTATEWITHHOLD	Payroll State Withholding	Payroll
REPO_PAYROLLW2	Form W-2 - Wage and Tax Statement	Payroll
REPO_PROJECT_ACCOUNTING	Project Accounting	
REPO_PURCHASEORDER	Purchase Order Reports	Purchase Orders
REPO_PURCHASES	Purchases	Accounting
REPO_QUOTA	Quota Reports	Sales Force Automation
REPO_RECONCILE	Reconcile Reporting	Accounting
REPO_ReminderEmployee	Employee Reminders	Employees
REPO_RETURNAUTH	Return Authorization Reports	Return Authorizations
REPO_REVREC	Revenue Recognition Reports	Revenue Recognition
REPO_RSRCALLOCATION	Resource Allocation Reports	Project Management
REPO_SALES	Sales	Sales Force Automation
REPO_SALESORDER	Sales Order Fulfillment Reports	Sales Orders
REPO_SALES_PARTNER	Sales By Partner	Partner Relationship Management
REPO_SALES_PROMO	Sales By Promotion Code	Sales Force Automation
REPO_SCHEDULE	Report Scheduling	Reports
REPO_SFA	Sales Force Automation	Customer Relationship Management
REPO_SUPPORT	Support	Customer Support and Service
REPO_TAX	Tax	Accounting
REPO_TIME	Time Tracking	Time Tracking
REPO_TRAN	Transaction Detail	Transactions
REPO_TRIALBALANCE	Trial Balance	Accounting

Permission ID	Permission Name	Feature
REPO_UNBILLED	Accounts Receivable Un-Billed	Accounting
REPO_WEBSITE	Web Site Report	Web Site
REPO_WEBSTORE	Web Store Report	Web Store
TRAN_ADJUSTMENTJOURNAL	Currency Adjustment Journal	Accounting
TRAN_ALLOCSCHEDULE	Create Allocation Schedules	Expense Allocation
TRAN_AMENDW4	Amend W-4	Employees
TRAN_APPROVECOMMISSN	Employee Commission Transaction Approval	Employee Commissions
TRAN_APPROVEDD	Approve Direct Deposit	Direct Deposit
TRAN_APPROVEEFT	Approve EFT	Electronic Funds Transfer
TRAN_APPROVEPARTNERCOMM	Partner Commission Transaction Approval	Partner Commissions/Royalties
TRAN_APPROVEVP	Approve Vendor Payments	ACH Vendor Payments
TRAN_AUDIT	Audit Trail	Transactions
TRAN_BILLPAY_APPROVE	Approve Online Bill Payments	Online Bill Pay
TRAN_BILLPAY_STATUS	View Online Bill Pay Status	Online Bill Pay
TRAN_BINTRNFR	Bin Transfer	Bin Management
TRAN_BINWKSHT	Bin Putaway Worksheet	Bin Management
TRAN_BLANKORD	Blanket Purchase Order	Purchasing and Receiving
TRAN_BLANKORDAPPRV	Blanket Purchase Order Approval	Purchasing and Receiving
TRAN_BUDGET	Set Up Budgets	Accounting
TRAN_BUILD	Build Assemblies	Assembly Items
TRAN_CARDCHRG	Credit Card	Accounting
TRAN_CASHRFND	Cash Sale Refund	Accounting
TRAN_CASHSALE	Cash Sale	Accounting
TRAN_CHARGE	Charge	Project Management
TRAN_CHARGERULE	Charge Rule	Project Management
TRAN_CHECK	Check	Accounting
TRAN_CLEARHOLD	Override Payment Hold	Order Management
TRAN_COMMISSN	Employee Commission Transaction	Employee Commissions
TRAN_COPY_BUDGET	Copy Budgets	Accounting
TRAN_CUSTCHRG	Statement Charge	A/R
TRAN_CUSTCRED	Credit Memo	A/R
TRAN_CUSTDEP	Customer Deposit	A/R
TRAN_CUSTINVC	Invoice	A/R

Permission ID	Permission Name	Feature
TRAN_CUSTINVCAPPRV	Invoice Approval	Order Management
TRAN_CUSTPYMT	Customer Payment	A/R
TRAN_CUSTRFND	Customer Refund	A/R
TRAN_DEPAPPL	Deposit Application	A/R
TRAN_DEPOSIT	Deposit	Accounting
TRAN_EDITPROFILE	Edit Profile	Employees
TRAN_ESTIMATE	Estimate	Estimates
TRAN_EXPREPT	Expense Report	Expense Reports
TRAN_FFTREQ	Fulfillment Request	Order Management
TRAN_FINCHRG	Finance Charge	A/R
TRAN_FIND	Find Transaction	Transactions
TRAN_FINDOLBMATCH	Find Matching Online Banking Transactions	Accounting
TRAN_FORECAST	Edit Forecast	Sales Force Automation
TRAN_FXREVAL	Currency Revaluation	Multiple Currencies
TRAN_GST_REFUND	Process GST Refund	Accounting
TRAN_IMPORTOLBFILE	Import Online Banking (QIF) File	Accounting
TRAN_INTERCOADJ	Intercompany Adjustments	Accounting
TRAN_INVADJST	Adjust Inventory	Inventory
TRAN_INVCOUNT	Count Inventory	Advanced Inventory Management
TRAN_INVDISTR	Distribute Inventory	Multi-Location Inventory
TRAN_INVREVAL	Revalue Inventory Cost	Inventory
TRAN_INVTRNFR	Transfer Inventory	Multi-Location Inventory
TRAN_INWKSHIT	Adjust Inventory Worksheet	Inventory
TRAN_ITEMRCPT	Receive Items	Advanced Receiving
TRAN_ITEMSHIP	Ship Items	Advanced Shipping
TRAN_JOURNAL	Make Journal Entry	Accounting
TRAN_JOURNALAPPRV	Journal Approval	Accounting
TRAN_LIABPYMT	Payroll Liability Payments	Payroll
TRAN_MGRFORECAST	Edit Manager Forecast	Sales Force Automation
TRAN_OLBSTATEMENT	Online Banking Statement	Accounting
TRAN_OPENBAL	Enter Opening Balances	Accounting
TRAN_OPRTNTY	Opportunity	Opportunities
TRAN_PARTNERCOMMISSN	Partner Commission Transaction	Partner Commissions/Royalties
TRAN_PAYCHECK	Individual Paycheck	Payroll

Permission ID	Permission Name	Feature
TRAN_PAYMENTAUDIT	Access Payment Audit Log	Order Management
TRAN_PAYROLLRUN	Process Payroll	Payroll
TRAN_PCHKJRNL	Paycheck Journal	Employees
TRAN_POSTVENDORBILLVARIANCE	Post Vendor Bill Variances	Vendors
TRAN_PRICELIST	Generate Price Lists	Item Record Management
TRAN_PRINT	Print/Email/Fax	Transactions
TRAN_PRINTCHECKSFORMS	Print Checks and Forms	Accounting
TRAN_PURCHCON	Purchase Contract	Vendors
TRAN_PURCHCONAPPRV	Purchase Contract Approval	Vendors
TRAN_PURCHORD	Purchase Order	Purchase Orders
TRAN_PURCHORDBILL	Bill Purchase Orders	Advanced Receiving
TRAN_PURCHORDRECEIVE	Receive Purchase Orders	Purchase Orders
TRAN_PURCHREQ	Requisition	Purchasing and Receiving
TRAN_PURCHREQAPPRV	Requisition Approval	Purchasing and Receiving
TRAN_QUOTA	Establish Quotas	Sales Force Automation
TRAN_RECOG_GIFTCERT_INCOME	Recognize Gift Certificate Income	Accounting
TRAN_RECONCILE	Reconcile	Accounting
TRAN_REVARRNG	Revenue Arrangement	Revenue Recognition
TRAN_REVARRNGAPPRV	Revenue Arrangement Approval	Revenue Recognition
TRAN_REVCOMM	Revenue Commitment	Revenue Commitments
TRAN_REVCOMRV	Revenue Commitment Reversal	Revenue Commitments
TRAN_REVCONTR	Revenue Contracts	Revenue Recognition
TRAN_RFQ	Request For Quote	Purchasing and Receiving
TRAN_RTNAUTH	Return Authorization	Return Authorizations
TRAN_RTNAUTHAPPRV	Return Auth. Approval	Return Authorizations
TRAN_RTNAUTHCREDIT	Refund Returns	Order Management
TRAN_RTNAUTHRECEIVE	Receive Returns	Order Management
TRAN_RTNAUTHREVERSEREVCOMMIT	Generate Revenue Commitment Reversals	Revenue Commitments
TRAN_SALESORD	Sales Order	Sales Orders
TRAN_SALESORDAPPRV	Sales Order Approval	Sales Orders
TRAN_SALESORDCOMMITREVENUE	Generate Revenue Commitment	Revenue Commitments
TRAN_SALESORDFULFILL	Fulfill Orders	Order Management
TRAN_SALESORDINVOICE	Bill Sales Orders	Advanced Shipping

Permission ID	Permission Name	Feature
TRAN_SALESORDREVENUECONTRACT	Generate Single Order Revenue Contracts	Revenue Recognition
TRAN_STATEMENT	Generate Statements	A/R
TRAN_STATUSDD	Direct Deposit Status	Direct Deposit
TRAN_STATUSEFT	EFT Status	Electronic Funds Transfer
TRAN_STATUSVP	Vendor Payment Status	ACH Vendor Payments
TRAN_STPICKUP	Store Pickup Fulfillment	SuiteCommerce InStore
TRAN_TAXLIAB	Pay Tax Liability	Accounting
TRAN_TAXPYMT	Pay Sales Tax	Accounting
TRAN_TEGPYBL	Tegata Payable	Accounting
TRAN_TEGRCVBL	Tegata Receivable	Accounting
TRAN_TIMEBILL	Track Time	Time Tracking
TRAN_TIMEPOST	Post Time	Project Management
TRAN_TRANSFER	Transfer Funds	Accounting
TRAN_TRNFRORD	Transfer Order	Inventory Management
TRAN_TRNFRORDAPPRV	Transfer Order Approval	Inventory Management
TRAN_UNBUILD	Unbuild Assemblies	Assembly Items
TRAN_VENDAUTH	Vendor Return Authorization	Vendor Return Authorizations
TRAN_VENDAUTHAPPRV	Vendor Return Auth. Approval	Vendor Return Authorizations
TRAN_VENDAUTHCREDIT	Credit Returns	Vendor Returns
TRAN_VENDAUTHRETURN	Vendor Returns	Vendor Return Authorizations
TRAN_VENDBILL	Bills	A/P
TRAN_VENDBILLAPPRV	Vendor Bill Approval	Vendor Bills
TRAN_VENDCRED	Enter Vendor Credits	A/P
TRAN_VENDPYMT	Pay Bills	A/P
TRAN_VENDRFQ	Vendor Request For Quote	Purchasing and Receiving
TRAN_WOCLOSE	Work Order Close	Inventory Management
TRAN_WOCOMPL	Work Order Completion	Inventory Management
TRAN_WOISSUE	Work Order Issue	Inventory Management
TRAN_WORKORD	Work Order	Work Orders
TRAN_WORKORDBUILD	Build Work Orders	Work Orders
TRAN_WORKORDCLOSE	Close Work Orders	Inventory Management
TRAN_WORKORDCOMPLETE	Enter Completions	Inventory Management
TRAN_WORKORDISSUE	Issue Components	Inventory Management
TRAN_WORKORDMARKBUILT	Mark Work Orders Built	Inventory Management
TRAN_WORKORDMARKFIRMED	Mark Work Orders Firmed	Inventory Management

Permission ID	Permission Name	Feature
TRAN_WORKORDMARKRELEASED	Mark Work Orders Released	Inventory Management
TRAN_YTDADJST	Enter Year-To-Date Payroll Adjustments	Payroll

# Feature Names and IDs



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table provides the internal IDs and feature names for all NetSuite features. You can use the feature ID with nlobjContext.getSetting(type, name) to see if a given feature is enabled in your account.

Feature Internal ID	Feature Name
ACCOUNTING	Accounting
ACCOUNTINGPERIODS	Accounting Periods
ACHVEND	ACH Vendor Payments
ADVANCEDJOBS	Project Management
ADVANCEDPRINTING	Advanced PDF/HTML Templates
ADVANCEDPROCUREMENTAPPROVALS	Advanced Procurement Approvals
ADVANCEDPROMOTIONS	Advanced Promotions
ADVANCEDREVENUERECOGNITION	Advanced Revenue Management
ADVANCEDSITECUST	Advanced Site Customization
ADVANCEDSITEMANAGEMENT	Site Management Tools
ADVBILLING	Advanced Billing
ADVBINSERIALLOTMGMT	Advanced Bin/Numbered Inventory Management
ADVFORECASTING	Advanced Forecasting
ADVINVENTORYMGMT	Advanced Inventory Management
ADVPARTNERACCESS	Advanced Partner Access
ADVRECEIVING	Advanced Receiving
ADVSHIPPING	Advanced Shipping
ADVTAXENGINE	Advanced Taxes
ADVWEBREPORTS	Advanced Web Reports
ADVWEBSEARCH	Advanced Web Search
ALTERNATIVEPAYMENTS	Alternative Payments
ALTSALESADVFORECAST	ALT_SALES Advanced Forecasting
ALTSALESAMOUNT	Alternate Sales Amount
AMORTIZATION	Amortization
APPROVALROUTING	Approval Routing
ASSEMBLIES	Assembly Items
ASYNCCUSTOMER	Asynchronous AfterSubmit Customer Processing
ASYNCSALESORDER	Asynchronous AfterSubmit Sales Order Processing
AUTOLOCATIONASSIGNMENT	Automatic Location Assignment

Feature Internal ID	Feature Name
AVAILABLETOPROMISE	Available To Promise
BARCODES	Bar Coding and Item Labels
BILLINGACCOUNTS	Billing Accounts
BILLINGCLASSES	Per-Employee Billing Rates
BILLINGWORKCENTER	Billing Operations
BILLSCOSTS	Bill Costs To Customers
BINMANAGEMENT	Bin Management
BLANKETPURCHASEORDERS	Blanket Purchase Orders
BOXNET	Box Document Management
CAMPAIGNASSISTANT	Campaign Assistant
CAMPAIGNSUBSCRIPTIONS	Subscription Categories
CCTRACKING	Credit Card Payments
CHARGEBASEDBILLING	Charge-Based Billing
CHECKOUTSUBDOMAIN	Customizable Checkout Subdomains
CLASSES	Classes
COMMISSIONONCUSTOMFIELDS	Commission on Custom Fields
COMMISSIONS	Employee Commissions
CONSOLPAYMENTS	Consolidated Payments
CREATESUITEBUNDLES	Create bundles with SuiteBundler
CRM	Customer Relationship Management
CRMTIME	Time Tracking for CRM
CRM_TEMPLATE_CATEGORIES	CRM Template Categories
CUSTOMCODE	Client SuiteScript
CUSTOMERACCESS	Customer Access
CUSTOMGLLINES	Custom GL Lines
CUSTOMRECORDS	Custom Records
CUSTOMSEGMENTS	Custom Segments
CUSTOMTRANSACTIONS	Custom Transactions
DEPARTMENTS	Departments
DIRECTDEPOSIT	Direct Deposit
DISTRIBUTIONRESOURCEPLANNING	Distribution Resource Planning
DOCUMENTPUBLISHING	Document Publishing
DOCUMENTS	File Cabinet
DOCUMENTSANTIVIRUS	Antivirus Scanning on File Cabinet Files

Feature Internal ID	Feature Name
DOWNLOADITEMS	Sell Downloadable Files
DROPSHIPMENTS	Drop Shipments & Special Orders
DUPLICATES	Duplicate Detection & Merge
DYNALLOCATION	Dynamic Allocation
EBAY	eBay Integration
EFT	Electronic Funds Transfer
EMAILINTEGRATION	Capture Email Replies
EMPPERMS	Global Permissions
ENHANCEDPREMIERPAYROLL	Enhanced Premier Payroll
ESCALATIONRULES	Automated Case Escalation
ESTIMATES	Estimates
EXPENSEALLOCATION	Expense Allocation
EXPREPORTS	Expense Reports
EXTCRM	Online Forms
EXTREMELIST	Inline Editing
EXTSTORE	External Catalog Site (WSDK)
FXRATEUPDATES	Currency Exchange Rate Integration
GIFTCERTIFICATES	Gift Certificates
GLAUDITNUMBERING	GL Audit Numbering
GRIDORDERMANAGEMENT	Grid Order Management
GROSSPROFIT	Gross Profit
GROUPAVERAGECOSTING	Group Average Costing
HELPDESK	Help Desk
HISTORICALMETRICS	Historical Metrics
I18NTAXREPORTS	International Tax Reports
INBOUNDCASEEMAIL	Email Case Capture
INTERCOMPANYAUTODROPSHIP	Automated Intercompany Drop Ship
INTERCOMPANYAUTOELIMINATION	Automated Intercompany Management
INTERCOMPANYTIMEEXPENSE	Intercompany Time and Expense
INTRANET	Intranet
INVENTORY	Inventory
INVENTORYCOUNT	Inventory Count
IPADDRESSRULES	IP Address Rules
ISSUEDB	Issue Management
ITEMDEMANDPLANNING	Demand Planning

Feature Internal ID	Feature Name
ITEMOPTIONS	Item Options
JOBCOSTING	Job Costing and Project Budgeting
JOBS	Projects
KNOWLEDGEBASE	Knowledge Base
KPIREPORTS	KPI Scorecards
LANDED COST	Landed Cost
LEADMANAGEMENT	Lead Conversion
LOCATIONS	Locations
LOTNUMBEREDINVENTORY	Lot Tracking
MAILMERGE	Mail Merge
MARKETING	Marketing Automation
MATRIXITEMS	Matrix Items
MFGROUTING	Manufacturing Routing and Work Center
MFGWORKINPROCESS	Manufacturing Work In Process
MOBILEPUSHNTF	Mobile Push Notification
MOSS	EU Mini One Stop Shop
MULTIBOOKMULTICURR	Multi-Book Accounting and Multiple Currencies
MULTICURRENCY	Multiple Currencies
MULTICURRENCYCUSTOMER	Multi-Currency Customers
MULTICURRENCYVENDOR	Multi-Currency Vendors
MULTILANGUAGE	Multi-Language
MULTILOCINVT	Multi-Location Inventory
MULTIPARTNER	Multi-Partner Management
MULTIPLEBUDGETS	Multiple Budgets
MULTIPLECALENDARS	Multiple Calendars
MULTISHIPTO	Multiple Shipping Routes
MULTISITE	Multiple Web Sites
MULTIVENDOR	Multiple Vendors
MULTPRICE	Multiple Prices
NETSUITEAPPROVALSWORKFLOW	NetSuite Approvals Workflow
NOTALTSALESAMOUNT	Not ALT_SALES Amount
NOTMULTIPARTNER	Not Multipartner
NOTTEAMSELLING	Not Team Selling
ONLINEORDERING	Online Ordering
OPENIDSSO	OpenID Single Sign-on

Feature Internal ID	Feature Name
OPPORTUNITIES	Opportunities
OUTLOOKINTEGRATION_V3	Outlook Integration
PARTNERACCESS	Partner Access
PARTNERCOMMISSIONS	Partner Commissions/Royalties
PAYABLES	A/P
PAYCHECKJOURNAL	Paycheck Journal
PAYPALINTEGRATION	PayPal Integration
PAYROLL	Payroll
PAYROLLSERVICE	Payroll Service
PICKPACKSHIP	Pick, Pack and Ship
PRM	Partner Relationship Management
PROJECTTASKMANAGER	Project Task Manager
PROMOCODES	Promotion Codes
PURCHASECARDATA	Send Purchase Card Data
PURCHASECONTRACTS	Purchase Contracts
PURCHASEORDERS	Purchase Orders
PURCHASEREQS	Purchase Requests
QUANTITYPRICING	Quantity Pricing
RECEIVABLES	A/R
REQUISITIONS	Requisitions
RESOURCEALLOCATIONAPPROVAL	Resource Allocation Approval Workflow
RESOURCEALLOCATIONCHART	Resource Allocation Chart
RESOURCEALLOCATIONS	Resource Allocations
RESOURCESKILLSETS	Resource Skill Sets
RETURNAUTHS	Return Authorizations
REVENUECOMMITMENTS	Revenue Commitments
REVENUERECOGNITION	Revenue Recognition
REVRECSALESORDERFORECASTING	Sales Order Revenue Forecasting
REVRECVSOE	VSOE
RFQ	Request For Quote
RUM	Real User Monitoring
SALESCAMPAIGNS	Sales Campaigns
SALESORDERS	Sales Orders
SAMLSSO	SAML Single Sign-on
SERIALIZEDINVENTORY	Serialized Inventory

Feature Internal ID	Feature Name
SERVERSIDESRIPTING	Server SuiteScript
SERVICEPRINTEDCHECKS	Service Printed Checks and Stubs
SERVICEPRINTEDW2S	Service Printed W-2s and 1099s
SFA	Sales Force Automation
SFA_AND_NOTASA	SFA And Not ALT_SALES Amount
SHIPPINGLABELS	Shipping Label Integration
SITELOCATIONALIASES	Descriptive URLs
SOFTDESCRIPTORS	Credit Card Soft Descriptors
STANDARDCOSTING	Standard Costing
STATACCOUNTING	Statistical Accounts
SUBSCRIPTIONBILLING	Subscription Billing
SUITEANALYTICSCONNECT	SuiteAnalytics Connect
SUITECOMMERCEENTERPRISE	SuiteCommerce Advanced
SUITESIGNON	SuiteSignOn
SUITESOCIAL	SuiteSocial
SUPPLYCHAINMANAGEMENT	Supply Chain Management
SUPPORT	Customer Support and Service
TABLEAU	Tableau Workbook Export
TAXAUDITFILES	Tax Audit Files
TBA	Token-based Authentication
TEAMSELLING	Team Selling
TELEPHONY	Telephony Integration
TIMETRACKING	Time Tracking
TRANDELETIONREASONCODE	Use Deletion Reason
UNITSOFMEASURE	Multiple Units of Measure
UPSELL	Upsell Manager
URLCOMPONENTALIASES	URL Component Aliases
VENDORACCESS	Vendor Access
VENDORRETURNAUTHS	Vendor Return Authorizations
WARRANTYANDREPAIRSMANAGEMENT	Warranty and Repairs Management
WEBAPPLICATIONS	SuiteScript Server Pages
WEBAPPLICATIONVERSIONING	SuiteScript Server Pages Versioning
WEBHOSTING	Host HTML Files
WEBSERVICESEXTERNAL	Web Services
WEBSITE	Web Site

Feature Internal ID	Feature Name
WEBSTORE	Web Store
WITHHOLDINGTAX	Withholding Tax
WORKFLOW	SuiteFlow
WORKORDERS	Work Orders

# Preference Names and IDs



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following tables list the internal IDs for all NetSuite preference configuration pages that support SuiteScript.

To set values on any of these pages in SuiteScript 1.0, you must first load the page using [nlapiLoadConfiguration\(type\)](#). After the page loads, you can then get or set all configuration values using the methods on the [nlobjConfiguration](#) object.

To set values on any of these pages in SuiteScript 2.0, you must first load the page using [config.load\(options\)](#). After the page loads, you can get or set all configuration values using the returned [record.Record](#) object. Additionally, you can use [User.getPreference\(options\)](#) to get the values for **General Preferences** and **Accounting Preferences**.

NetSuite configuration preference IDs are grouped into the following categories:

- General Preferences
- Company Information
- User Preferences
- Accounting Preferences
- Accounting Periods
- Tax Setup
- Tax Periods

## General Preferences

These are the account preferences that can be found by going to Setup > Company > General Preferences.

The internal ID for the General Preferences page is **companypreferences**. All preference internal IDs are case-insensitive.

Preference UI Label	Preference Internal ID
Use State Abbreviations in Addresses	ABBREVIATESTATES
Assign Tasks to Partners	ASSIGNTASKSTOPARTNERS
Auto Name Customers	AUTONAMECUSTOMERS
Calendar System	CALENDARSYSTEM
Company Logo Folder	COMPANYLOGOFOLDER
Add Primary Contact to Bill To Address	CONTACTONBILLTO
Default Role for New Customers	CUSTOMERROLE
Default Customer Type	CUSTOMERTYPE

Preference UI Label	Preference Internal ID
Customer Center Welcome Message	CUSTOMERWELCOMEMESSAGE
Date Format	DATEFORMAT
Delay Loading of Sublists	DELAYLOADINGSUBLISTS
Default Partner Type	DFLTPARTNERTYPE
Default Vendor Type	DFLTVENDORTYPE
Email Employee on Approvals	EMAILEMPLOYEEONAPPROVAL
Maintenance Complete Email Notification	EMAILMAINTENANCECOMPLETE
First Day of Week	FIRSTDAYOFWEEK
Screen Font	FONT
Allow Free-Form States in Addresses	FREEFORMSTATES
Hide Attachment Folders	HIDEATTACHMENTFOLDERS
Internal Web Site	INTERNALWEBSITE
Show Display Name with Item Codes	ITEMNUMBERING
Use Last Name First for Employees	LASTNAMEFIRST
Use Last Name First for Entities	LASTNAMEFIRSTENTITIES
Number of rows in List segments	LISTSEGMENTSIZE
Long Date Format	LONGDATEFORMAT
Maximum entries in Dropdown	MAXDROPDOWNSIZE
Maximum number of dimension columns allowed in a report	MAXREPORTDIMENSIONS
Maximum number of rows allowed in a report	MAXREPORTROWS
Minimum Password Length	MINPASSWORDLENGTH
Password Expiration in Days	PASSWORDEXPIREDAYS
Phone Number Format	PHONEFORMAT
Pre-Populate Contact Address	PREPOPULATECONTACTADDRESS
Show Reports in Grid	REPORTGRID
Show Employees as Contacts	SHOWEMPLOYEESASCONTACTS
Show Individuals as Contacts	SHOWINDIVIDUALSASCONTACTS
Show List When Only One Results	SHOWLISTONRESULT
Show Page Feedback Link	SHOWPAGEFEEDBACKLINK
Show Quick Add Row on Lists	SHOWQUICKADD
Time Format	TIMEFORMAT

## Company Information

These are the account preferences that can be found by going to Setup > Company > Company Information.



The internal ID for the Company Information page is **companyinformation**. All preference internal IDs are case-insensitive.



**Important:** The fields listed below do not correspond exactly to the fields displayed on the Company Information page as of Version 2014 Release 2. The enhanced address customization supported as of this release resulted in changes to the Company Information UI. To preserve backwards compatibility of pre-existing scripts, the preference internal IDs used for scripting with the companyinformation configuration object were not changed. You should continue to use the IDs listed below when scripting with companyinformation.

Preference Label	Preference Internal ID
Company Name	companyname
Legal Name	legalname
Ship to Attention	attention
Address 1	address1
Address 2	address2
City	city
County/State/Province	state
Zip	zip
Country	country
Address	addresstext
Employer Identification Number (EIN)	employerid
SSN or TIN (Social Security Number, Tax ID Number)	taxid
Return Email Address	email
Phone	phone
Fax	fax
Web Site	url
Company Logo (Forms)	formlogo
Company Logo (Pages)	pagelogo
	purgeaccount
Display Logo Internally	displaylogointernal
First Fiscal Month	fiscalmonth
Time Zone	timezone
Currency	basecurrency
<b>For shipping addresses:</b>	
Address1	shippingaddress1
Address2	shippingaddress2
City	shippingcity
County/State/Province	shippingstate
Zip	shippingzip

Preference Label	Preference Internal ID
Country	shippingcountry
Address	shippingaddressstext
<b>For return addresses:</b>	
Address1	returnaddress1
Address2	returnaddress2
City	returncity
County/State/Province	returnstate
Zip	returnzip
Country	returncountry
Address	returnaddressstext

## User Preferences

These are the user preferences that can be found by going to Home > Set Preferences.

The internal ID for the user preferences page (which appears as the Set Preferences page in the UI) is `userpreferences`. All preference internal IDs are case-insensitive.

Be aware that the API for setting user preferences works the same as the UI in terms of setting a preference permanently or just for a user's session. In the UI if a user sets a preference, and the preference reverts back to a default setting on the user's next login, the same behavior is supported in SuiteScript.

Preference UI Label	Preference Internal ID
<b>On the General tab</b>	
Nickname	MESSAGE_NICKNAME
Signature	MESSAGE_SIGNATURE
Add Signature to Messages	MESSAGE_AUTOSIGNATURE
From Email Address	MESSAGE_EMAIL
Language	LANGUAGE
PDF Language	PDFLANGUAGE
Time Zone	TIMEZONE
Date Format	DATEFORMAT
Long Date Format	LONGDATEFORMAT
Time Format	TIMEFORMAT
Number Format	NUMBERFORMAT
Negative Number Format	NEGATIVE_NUMBER_FORMAT
Phone Number Format	PHONEFORMAT
Auto Place Decimal	AUTOPLACE
Use Multicurrency Expense Reports	USE_MC_ON_EXPREPT



Preference UI Label	Preference Internal ID
Download PDF Files	DOWNLOADPDFS
Address Mapping Type	MAPTYPE
Show Internal IDs	EXPOSEIDS
Only Show Last Subaccount	ONLYSHOWLASTSUBACCT
Only Show Last Subentity	ONLYSHOWLASTSUBENT
Only Show Last Subitem	ONLYSHOWLASTSUBITEM
Submit Warnings	SUBMITWARNINGS
Limit CC Field to Contacts & Employees	EMAILLIMITCC
Default Issue Email Notification	ISSUE_EMAIL_ME_WHEN
Notify Me Upon Issue Assignment	ISSUE_NOTIFY_UPON_ASSIGNMENT
Cache Pages in Browser	CACHEPAGES
Delay Loading of Sublists	DELAYLOADINGSUBLISTS
Number of Rows in List Segments	LISTSEGMENTSIZE
Maximum Entries in Dropdowns	MAXDROPOUTSIZE
Type-Ahead on List Fields	TYPEAHEADSELECTS
Require Exact Match on Item Type-Ahead	ITEMEXACTMATCH
Show Quick Add Row on Lists	SHOWQUICKADD
<b>On the Appearance tab</b>	
Screen Font	FONT
Compensate for Large Fonts	SYSTEMLARGEFONTS
Register Look on Lists	REGISTERSTYLE
Only Show Field Boarders on Hover	SHOWFIELDBORDERONHOVER
Chart Theme	CHART_THEME
Chart Background	CHART_BACKGROUND
Use Classic Interface	BASICCENTER
Landing Page	LANDINGPAGE
Show Portlet Hint	SHOWPORTLETHINT
Limit Entry Forms to Two Columns	LIMITTOTWOCOLUMNS
Expand Tabs on Entry Forms	UNLAYEREDTABS
Enable Rich Text Editing	RICHTEXTEDITOR
Default Rich Text Editor Font	EDITORFONT
Default Rich Text Editor Font Size	EDITORFONTSIZE
<b>On the Transactions tab</b>	
Auto Fill Transactions	AUTOFILL
Transaction Email Attachment Format	TRANSACTION_ATTACHMENT_FORMAT



Preference UI Label	Preference Internal ID
Alphabetize Items Regardless of Type	ALPHABETIZE_ITEMS
Duplicate Number Warnings	DUPLICATEWARNINGS
Inventory Level Warnings	STOCKWARNINGS
Customer Credit Limit Handling	CUSTCREDLIMHANDLING
Vendor Credit Limit Warnings	VENDCREDLIMWARNINGS
Print Using HTML	HTMLPRINTING
Email Using HTML	HTMLEMAIL
Horizontal Print Offset	HORZPRINTOFFSET
Vertical Print Offset	VERTPRINTOFFSET
<b>On the Reporting/Search tab</b>	
Report by Period	REPORTBYPERIOD
Print Company Logo	DISPLAYLOGO
Display Report Title on Screen	DISPLAYRPTTITLE
Display Report Description	DISPLAYRPTDESC
Default Bank Account	DEFAULT_BANKREG
Show Forecasts as Weighted	FORECASTWEIGHTED
Show List When Only One Result	SHOWLISTONERESULT
Quick Search Uses Keywords	KEYWORDSEARCH
Popup Search Uses Keywords	KEYWORDSEARCHPOPUP
Include Inactives in Global & Quick Search	SEARCHINACTIVES
Popup Auto Suggest	POPUPAUTOSUGGEST
Global Search Auto Suggest	SEARCHAUTOSUGGEST
Global Search Sort by Name/ID	GLOBALSEARCHSORTBYNAME
Global Search Customer Prefix Includes Leads and Prospects	GLOBALSEARCHCUPREFIX
PDF Page Orientation	REPORTPDFORIENTATION
PDF Font Size	REPORTPDFFONTSIZE
CSV Export Character Encoding	CSVEXPORTENCODING
<b>On the Activities tab</b>	
Edit Activities from Calendar	EVENT_EDITFROMCALENDAR
Send Invitation Emails	EVENT_EMAILNOTIFICATION
Restrict Invitees to Employees	EVENT_INTERNALINVITEESONLY
Default Event Access Setting for New Events	EVENT_DEFAULTPUBLIC
Default Reminder Type	REMINDERTYPE
Default Reminder Time	REMINDERPERIOD
Play Audio with Popup Event Reminders	REMINDERPLAYWAVE

Preference UI Label	Preference Internal ID
Default Priority for Tasks	DEFAULTTASKPRIORITY
Default New Tasks Public	TASK_DEFAULTPUBLIC
Default New Phone Calls Public	CALL_DEFAULTPUBLIC
Default Sync Category	DEFAULT_CONTACT_SYNC_CATEGORY
<b>On the Alerts tab</b>	
First Selection	EMAILALERT_AM
Second Selection	EMAILALERT_NOON
Third Selection	EMAILALERT_PM
Include links in HTML alerts	LINKS_EMAILALERT
Respect Quick Date Portlet Settings	USE_QUICKDATE_IN_ALERTS
E-Mail	EMAILALERT_EMAIL
Send an On-Demand Alert from this Role	ALERTONDemand
<b>On the Restrict View tab</b>	
Subsidiary	SUBSIDIARY
Include Sub-Subsidiaries	SUBSIDIARYSUBS
Department	DEPARTMENT
Include Sub-Departments	DEPARTMENTSUBS
Include Unassigned	DEPARTMENTUNASSIGNED
Location	LOCATION
Include Sub-Locations	LOCATIONSUBS
Include Unassigned	LOCATIONUNASSIGNED
Class	CLASS
Include Sub-Classes	CLASSTSUBS
Include Unassigned	CLASSUNASSIGNED
<b>On the Telephony tab</b>	
Telephony Option	TELEPHONY_OPTION
TAPI Device	TELEPHONYDEVICE
CTI URL	CTI_URL
Prefix to Dial Out	DIALOUTPREFIX
Send Notifications When	ISSUE_NOTIFICATION
Send Notifications To	ISSUE_NOTIFICATION_EMAILS

## Accounting Preferences

These are the account preferences that can be found by going to Setup > Accounting > Accounting Preferences. All preference internal IDs are case-insensitive.

The internal ID for the Accounting Preferences page is **accountingpreferences**.

Preference UI Label	Preference Internal ID
<b>On the General tab</b>	
Use Account Numbers	ACCOUNTNUMBERS
Aging Reports Use	AGEFROM
Allow cross-subsidiary billable time and expenses	ALLOWCROSSSUBBILLABLES
Allow manual entry of Gift Certificate Codes	ALLOWMANUALGCCODE
Allow subsidiary hierarchy to be modified	ALLOWSUBSIDHIERARCHYCHANGE
Default Amortization Journal Date to	AMORJOURNALDATEDEFAULT
Expand Account Lists	EXPANDACCOUNTLISTS
Revenue Recognition/Adv. Billing: Use Sales Order Amount	CALCPCTCOMPTFROMSALESORDERAMT
Cash Basis Reporting	CASHBASIS
Always Allow Per-line Classifications on Journals	CDLPERLINEONJE
Allow Per-Line Classes	CLASSESPERLINE
Make Classes Mandatory	CLASSMANDATORY
Accept Payments through Top-level Customer	CONSOLPAYMENTS
Days Overdue for Warning/Hold	CREDLIMDAYS
Display Current Count on Adjustments	CURCOUNTADJUSTMENTS
Display Current Count on Transfers	CURCOUNTONTRANSFERS
Display Current Count on Worksheets	CURCOUNTONWORKSHEETS
Customer Credit Limit Handling	CUSTCREDITLIMHANDLING
Customer Credit Limit Includes Orders	CUSTCREDLIMORDERS
Make Departments Mandatory	DEPTMANDATORY
Allow Per-Line Departments	DEPTSPERLINE
Require Approvals on Journal Entries	JOURNALAPPROVALS
Make Locations Mandatory	LOCMANDATORY
Allow Per-Line Locations	LOCSPERLINE
Maximum number of MLI locations	MAXLOCATIONS
Maximum number of Subsidiaries	MAXSUBSIDIARIES
Allow Users to Modify Amortization Schedule	MODIFYAMORTOTALAMOUNT
Allow Users to Modify Revenue Recognition Schedule	MODIFYREVRECTOTALAMOUNT
Allow Users to Modify VSOE Values on Transactions	MODIFYVSOEVALSONTRAN
Name for Tax Amount	NAMINGTAXAMOUNT
Name for Tax Rate	NAMINGTAXRATE
Name for Tax Reg. Number	NAMINGTAXREGNUMBER
Allow Non-balancing Classifications on Journals	NONBALANCINGCDLONJE
Allow Empty Classifications on Journals	NULLCDLONJE

Preference UI Label	Preference Internal ID
Show Only Open Transactions on Statements	OPENONLYSTMTS
Prorate Revenue Recognition Dates For Partially Billed Sales Orders	PROPRATEREVRECINVFROMSO
Restrict Account Balance Viewing for Employees with Classification Restrictions	RESTRICTBALANCEVIEWING
Void Transactions Using Reversing Journals	REVERSALVOIDING
Invoice Revenue Recognition Dates Source from	REVRECDATESUPDATEMETHOD
Default Revenue Recognition Journal Date to	REVRECJOURNALDATEDEFAULT
Show Journal Memos on Statements	STATEMENTJOURNALMEMOS
Include Shipping for Term Discounts	TERMDISCOUNTSINCLUDESHIPPING
Include Tax for Term Discounts	TERMDISCOUNTSINCLUDETAX
Allow Revenue Commitment Reversals In Advance of Item Receipt	UNRECEIVEDREVENUECOMMITMENTS
Allow Revenue Commitments In Advance of Fulfillment	UNSHIPPEDREVENUECOMMITMENTS
Use System Calculated Percentage of Completion For Revenue Recognition/Amortization	USESYSCALCPCT4REVREC
Vendor Credit Limit Includes Orders	VENDCREDLIMORDERS
Vendor Credit Limit Warnings	VENDCREDLIMWARNINGS
Default Vendor Payments To Be Printed	VENDPYMTTOPRINT
<b>On the Items/Transactions tab</b>	
Allow Purchase of Assembly Items	ALLOWASSEMBLYPURCHASE
Default Asset Account	ASSETACCOUNT
Use Credit Card Security Code for Credit Card Transactions	CCSECURITYCODE
Centralize Purchasing in a Single Location	CENTRALIZEDPURCHASING
Default COGS Account	COGSACCOUNT
Consolidate Jobs on Sales Transactions	CONSOLINVOICES
Duplicate Number Warnings	DUPLICATEWARNINGS
Anyone Can Set Item Accounts	EDITITEMACCOUNTS
Default Estimate Expiration (in days)	ESTIMATEEXPIRATION
Use the Exact Cost for Linked Returns	EXACTCOSTONLINELINKEDRETURNS
Default Expense Account	EXPENSEACCOUNT
Customers Can Pay Online	EXTERNALPAYMENTS
Include Reimbursements in Sales and Forecast Reports	FORECASTINCLUDES_REIMB_EXP
Include Shipping in Sales and Forecast Reports	FORECASTINCLUDES_SHIPPING
Transaction Types to Exclude from Forecast Reports	FORECASTTRANTRYPES
Gift Certificate Auth Code Generation	GIFTCERTAUTHCODEGENERATION
Default Income Account	INCOMEACCOUNT

Preference UI Label	Preference Internal ID
Inventory Costing Method	INVTCOSTMETHOD
Days Before Lot Expiration Warning	LOTEXPIRATIONWARNING
Default Payment Account	PAYMENTACCOUNT
Payment Due Preference	PAYMENTDUEPREFERENCE
Purchase Discount Account	PURCHDISCACCT
Maximum # of Quantity-based Price Levels	QTYPRICECOUNT
Allow Quantity Discounts per Price Level on Schedules	QTYPRICESCHEDULEMULTDISCOUNTS
Sort Reconcile By	RECONSORTCOL
Require Bins on All Transactions Except Item Receipts	REQUIREBINSONTRANS
Sales Discount Account	SALESDISCACCT
Transaction Types to Exclude from Sales Reports	SALESTRANTYPES
Scan Individual Items	SINGLEITEMBARCODING
Matrix Item Name/Number Separator	SKUSEPARATOR
Tegata Maturity Date	TEGATAMATURITY
Recalculate Estimated Cost on Creation of Linked Transactions	USELATESTCOSTESTIMATE
Use Preferred Bin on Item Receipts	USEPREFERREDBINONITEMRCPT
Use Popup to Select Serial/Lot Numbers on Sales	USESERIALNUMBERSELECT
Use Zero Cost for Linked Underwater Sales	ZEROCOSTUNDERWATER
<b>On the Order Management tab</b>	
Build Based on Commitment	BUILDCOMMITTED
Filter Bulk Fulfillment Page by Location	BULKFULFILLOCFILTERING
Convert Absolute Discounts to Percentage When Billing	CONVERTABSOLUTEDISSCOUNTS
Default Location for Purchase Orders	DEFAULTPURCHASEORDERLOCATION
Default Location for Sales Orders	DEFAULTSALESORDERLOCATION
Default Items to Zero Received/Fulfilled	DEFAULTUNFULFILLED
Default Return Auth. Status	DEFRTNAUTHSTATUS
Default Sales Order Status	DEFALESORDSTATUS
Default Vendor Return Auth. Status	DEFVENDAUTHSTATUS
Include Committed Quantities on Drop Shipments/Special Orders	DROPSHIPINCLUDECOMMITTED
Drop Ship P.O. Form	DROPSHIPTEMPLATE
Send Email Confirmation when Sales Order Canceled	EMAILCANCELORDER
Automatically Email Drop Ship P.O.s	EMAILDROPSHIPPOS
Automatically Fax Drop Ship P.O.s	FAXDROPSHIPPOS
Fulfill Based on Commitment	FULFILLCOMMITTED

Preference UI Label	Preference Internal ID
Base Invoice Date on Billing Schedule Date	INVOICEUSESCHEDULEDATE
Limit Vendor List on Items	LIMITITEMVENDORS
Name for Packed Status	NAMINGPACKED
Name for Picked Status	NAMINGPICKED
Name for Shipped Status	NAMINGSHIPPED
Send Order Fulfilled Confirmation Emails	ORDFULFILLCONFEMAIL
Use Web Site Template for Fulfillment Emails	ORDFULFILLUSESTORETEMPLATES
Allow Overage on Assembly Builds	OVERBUILDS
Allow Overage on Item Fulfillments	OVERFULFILLMENTS
Allow Overage on Item Receipts	OVERRECEIPTS
Show Drop Ship Items on Packing Slips	PACKINGSLIPDROPSHIP
Limit Status on Packing Slip Queue	PACKINGSLIPSTATUS
Always Print Kit Items on Picking Tickets	PICKINGTICKETKITITEMS
Show Non-Inventory Items on Picking Tickets and Packing Slips	PICKINGTICKETNONINV
Show Uncommitted Items on Picking Tickets	PICKINGTICKETUNCOMMITTED
Allow Expenses on Purchase Order	POEXPENSES
Queue Drop Ship P.O.s for Printing	PRINTDROPSHIPPOS
Require Re-approval on Edit of Sales Order	REAPPROVESOONEDIT
Restock Returned Items	RESTOCKRETURNS
Show All Ordered Items on Packing Slips	SHOWOPENPACKSLIP
Show Unfulfilled Items on Invoices	SHOWUNSHIPPEDITEMS
Bill in Advance of Receipt	UNC RECEIVEDBILLS
Refund in Advance of Return	UNC RECEIVEDRTNAUTHS
Credit in Advance of Vendor Return	UNRETURNEDVENDAUTHS
Invoice in Advance of Fulfillment	UNSHIPPEDINVOICES
Write-Off Account for Returns	WRITEOFFACCOUNT
<b>On the Time &amp; Expenses tab</b>	
Automatically Notify Supervisor	AUTONOTIFYSUPV
Combine Detail Items on Expense Reports	COMBINEEXPENSEITEMS
Copy Expense Memos to Invoices	COPYEXPENSEMEMOS
Copy Time Memos to Invoices	COPYTIMEMEMOS
Expenses Billable by Default	DEFAULTEXPENSEBILLABLE
Items Billable by Default	DEFAULTITEMSBILLABLE
Time Billable by Default	DEFAULTTIMEBILLABLE
Override Rates on Time Records	OVERRIDETIMERATES

Preference UI Label	Preference Internal ID
Show Planned Time in Time Entry	SHOWPLANNEDTIME
Require Approvals on Time Records	TIMEAPPROVALS
Show Jobs Only for Time and Expense Entry	TIMEEXPENSEJOBONLY

## Accounting Periods

These are the account preferences that can be found by going to Setup > Accounting > Manage G/L > Manage Accounting Periods.

The internal ID for the Accounting Periods page is **accountingperiods**. All preference internal IDs are case-insensitive.

Preference UI Label	Preference Internal ID
First Fiscal Month	fiscalmonth
Fiscal Year End	fiscalyear
Period Format	periodstyle
Year in Period Name	periodnameyear
One-Day Year-End Adjustment Period	lastday

## Tax Setup

The internal ID for the Set Up Taxes page is **taxpreferences**. All preference internal IDs are case-insensitive.

Preference IDs for fields on this page vary according to the country of the nexus. Field internal IDs are suffixed with the nexus country code. The format for these preferences is <field internal id><nexus country code>. Be aware that different fields are available for different nexuses.

The table below shows some scriptable tax preference fields for a US nexus. Fields are suffixed with **us**, for the US nexus. This table is provided for example purposes.

Preference UI Label	Preference Internal ID
	defaulttaxableus
	type
	chargoutofdistrictus
	perlinetaxesus
	storeordertaxationus
	enabletaxlookupus

Field IDs in your account may be suffixed with a different nexus country code. And different fields may be available. You may be able to look up field IDs in the user interface, by going to Setup > Accounting > Set Up Taxes, and clicking on a nexus.

- To make field IDs available, go to Home > Set Preferences and ensure that the Show Internal IDs box is checked on the General subtab, Defaults area.
- Find the field in the NetSuite user interface and click the field label to display the field level help text. The field ID is displayed in the popup.

## Tax Periods

These are the tax preferences that can be found by going to Setup > Accounting > Taxes > Manage Tax Periods.

The internal ID for the Tax Periods page is **taxperiods**. All preference internal IDs are case-insensitive.

Preference UI Label	Preference Internal ID
First Fiscal Month	fiscalmonth
Fiscal Year End	fiscalyear
Period Format	periodstyle
Year in Period Name	periodnameyear

# Supported File Types

This section provides a list of all files types that can be defined in the *type* argument in the following APIs:

- [nlapiCreateFile\(name, type, contents\)](#) in SuiteScript Functions .
- [nlobjResponse.setContentType\(type, name, disposition\)](#) in SuiteScript Objects

When referencing a file type in the *type* argument, use the **file type ID**. See the following example:

```
nlapiCreateFile('helloworld.txt', 'PLAINTEXT', 'Hello World\nHello World');
```



**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.

File Type ID	Name	Extension	Content Type
AUTOCAD	AutoCad	.dwg	application/x-autocad
BMPIMAGE	BMP Image	.bmp	image/x-xbitmap
CSV	CSV File	.csv	text/csv
EXCEL	Excel File	.xls	application/vnd.ms-excel
FLASH	Flash Animation	.swf	application/x-shockwave-flash
GIFIMAGE	GIF Image	.gif	image/gif
GZIP	GNU Zip File	.gz	application/x-gzip-compressed
HTMLDOC	HTML File	.htm	text/html
ICON	Icon Image	.ico	image/ico
JAVASCRIPT	JavaScript File	.js	text/javascript
JPGIMAGE	JPEG Image	.jpg	image/jpeg
JSON	JSON File	.json	application/json
MESSAGERFC	Message RFC	.eml	message/rfc822
MP3	MP3 Audio	.mp3	audio/mpeg
MPEGMOVIE	MPEG Video	.mpg	video/mpeg
MSPPROJECT	Project File	.mpp	application/vnd.ms-project
PDF	PDF File	.pdf	application/pdf
PJPEGIMAGE	PJPEG Image	.pjpeg	image/pjpeg
PLAINTEXT	Plain Text File	.txt	text/plain
PNGIMAGE	PNG Image	.png	image/x-png
POSTSCRIPT	PostScript File	.ps	application/postscript
POWERPOINT	PowerPoint File	.ppt	application/vnd.ms-powerpoint
QUICKTIME	QuickTime Video	.mov	video/quicktime
RTF	RTF File	.rtf	application/rtf

File Type ID	Name	Extension	Content Type
SMS	SMS File	.sms	application/sms
STYLESHEET	CSS File	.css	text/css
TIFFIMAGE	TIFF Image	.tiff	image/tiff
VISIO	Visio File	.vsd	application/vnd.visio
WORD	Word File	.doc	application/msword
XMLDOC	XML File	.xml	text/xml
ZIP	Zip File	.zip	application/zip

# Button IDs



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists the internal IDs for standard NetSuite buttons that support SuiteScript. When using SuiteScript to rename or hide buttons, you will use the nlobjButton methods `setLabel(label)` and `setVisible(visible)`, respectively.

On some records, it is possible that certain buttons will appear as actions in the More Actions menu. You can still use the nlobjButton methods to change the labels of these actions and to hide or show the actions in the menu. (For information on the More Action menu, see *More Actions Menu* in the NetSuite Help Center.)



**Important:** Customizing the Save, Edit, Cancel, Back, and Reset buttons is not supported in SuiteScript or in point-and-click customization.



**Note:** Also note that you cannot use SuiteScript to change the display of an inline button to an action in the More Actions menu. Similarly, you cannot use SuiteScript to display an action as an inline button. To change the display type of buttons and actions, you must use SuiteBuilder point-and-click customization. See the help topic [Configuring Buttons and Actions](#) for details.

Button UI Label	Button Internal ID
Add Items	addmatrix
Accept	accept
Accept Payment	acceptpayment
Apply	apply
Approve	approve
Approve Return	approvereturn
Authorize Return	return
Auto Fill	autofill
Bill	bill
Bill Remaining	billremaining
Cancel Order	cancelorder
Cancel Return	cancelreturn
Clear Splits	clearsplits
Close	closeremaining
Convert	convertlead
Convert to Inventory	convertinvt
Convert to Lot Numbered Inventory	convertlot
Convert to Serialized Inventory	convertserial
Create Build	createbuild
Create Matrix	creatematrix

Button UI Label	Button Internal ID
Credit	credit
Decline	decline
Delete	delete
Email	email
Fax	fax
Fulfill	process
Generate Price List	generatepricelist
Generate Statement	generatestatement
GL Impact	gimpact
Go To Register	gotoregister
Grab	grab
Make Copy	makecopy
Make Payment	payment
Make Standalone Copy	makestandalonecopy
Memorize	memorize
Merge	merge
New	new
New Event Field	neweventfield
Next Bill	nextbill
Next Week	next
Prev Week	prev
Print	print
Print Bill of Materials	printbom
Print Label	printlabel
Print Labels	printlabels
Print Picking Ticket	printpicktick
Print Summary	depositsummary
Quick Accept	quickaccept
Recalc	recalc
Receive	receive
Refund	refund
Reject	reject
Renew	renewal
Reset	resetter
Revenue Commitment Reversal	revcomrv

Button UI Label	Button Internal ID
Save As	submitas
Save & Bill	submitbill
Save & Convert	submitconvert
Save & Copy	submitcopy
Save & Edit	submitedit
Save & Email	saveemail
Save & Fulfill	submitfulfill
Save & New	submitnew
Save & Next	submitnext
Save & Print	saveprint
Save & Print BOM	saveprintbom
Save & Print Label	saveandprintlabel
Save & Refund	submitrefund
Save & Same	submitsame
Save Baseline	savebaseline
Search	search
Show Activity	showactivity
Submit Invoice	submitinvoice
Tentative	tentative
Unbuild	createunbuild
Update Matrix	updatematrix
Update VSOE	updatevsoe
View All Transactions	viewalltransactions
Void	void
W4 Worksheet	w4data

# Supported Tasklinks



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists NetSuite tasklinks that can be referenced when using `nlapiResolveURL(type, identifier, id, displayMode)` or `nlapiSetRedirectURL(type, identifier, id, editmode, parameters)`.

Tasklinks are only available if the related feature is enabled in your account. Also, the following table lists only standard tasklinks. This list does not include tasklinks related to custom objects that have been created in your account. This list also does not include tasklinks related to objects that have been added to your account as part of SuiteApp installations.

Task ID	Page Label in NetSuite	URL
ADMI_ACCOUNTMOBILITY	Account Mobility Tasks	/app/setup/accountmobilitytasks.nl
ADMI_ACCTSETUP	Accounting Preferences	/app/setup/acctsetup.nl
ADMI_ACHSETUP	Set Up ACH Processing	/app/setup/achsetup.nl
ADMI_ACTIVATEBANKACCOUNT	Activate Bank Account	/app/payroll/verifyacct.nl
ADMI_ADVINVENTORYSETUP	Inventory Management Preferences	/app/setup/advinventorysetup.nl
ADMI_ALLOW_LOGIN	NetSuite Support Login	/app/crm/support/allowssupportlogin.nl
ADMI_ALTERNATIVEPAYMENTS	Alternative Payments	/app/setup/altpaymentaccounts.nl
ADMI_BACKUP	Full CSV Export	/app/external/export/backup/backup.nl?csv=T
ADMI_BASSETUP	Set Up Business Activity Statement	/app/setup/bassetup.nl
ADMI_BILLINGEVENTS	View Billing Information	/app/billing/billingevents.nl
ADMI_BILLINGTERMS	View Billing Terms	/app/billing/billingterms.nl
ADMI_BOUNCEDADDRESS	Bounced Email Addresses	/app/common/entity/bouncedaddresses.nl
ADMI_BUDGET	Import Set Up Budget	/app/setup/assistants/nsimport/simpleimport.nl?rectype=BUDGETIMPORT
ADMI_BUNDLEDETAILS	Bundle Details	/app/bundler/bundledetails.nl
ADMI_CLASSESTOLOCSS	Convert Classes to Locations	/app/setup/clasestolocs.nl
ADMI_CLEARACCOUNT	Delete All Data	/pages/setup/clearaccount.jsp
ADMI_CLOSEPERIOD	Lock Transactions	/app/setup/closeperiod.nl
ADMI_COMMERCECATEGORY	Admin Commerce Categories	/app/platform/services/commercecategory/commercecategories.nl
ADMI_COMMISSIONSETUP	Set Up Commissions	/app/setup/commissionsetup.nl

Task ID	Page Label in NetSuite	URL
ADMI_COMPANY	Company Information	/app/common/otherlists/subsidiarytype.nl?forcompany=T
ADMI_CONVERTCLASSES	Convert Classes to Departments	/app/setup/convertclasses.nl
ADMI_COPYCOA	Copy Chart of Accounts to New Company	/app/signup/consolidation.nl?coa=
ADMI_CREDCARD	Set Up Credit Card Processing	/app/setup/ccprocessorlist.nl
ADMI_CSIMPORTPREF	CSV Import Preferences	/app/setup/upload/csv/csvpreferences.nl
ADMI_CUSTAPPTEXT	Customize Fulfillment Email	/app/setup/customizetext.nl?scope=APP
ADMI_DUPLICATESETUP	Set Up Duplicate Detection	/app/setup/duplicatedetectsetup.nl
ADMI_EDITNEXUSES	New Nexus	/app/setup/nexus.nl
ADMI_EDITTAXACCTS	New Tax Control Account	/app/setup/taxacct.nl
ADMI_EDITTAXTYPES	New Tax Types	/app/setup/taxtype.nl
ADMI_EDITTEGATA	New Tegata Account	/app/setup/tegataaccount.nl
ADMI_EMAIL	Email Preferences	/app/setup/emailpreferences.nl
ADMI_EXPORTIIF	Export as IIF	/pages/setup/exportiiform.jsp
ADMI_EXPORTPRODUCTFEEDS	Product Feeds	/app/site/setup/exportproductfeeds.nl
ADMI_EXPORTPRODUCTFEEDSADV	Product Feeds	/app/site/setup/exportproductfeeds.nl
ADMI_FEATURES	Enable Features	/app/setup/features.nl
ADMI_FEDEXREG	Fedex Registration Wizard	/app/common/shipping/fedex/fedexregistration.nl
ADMI_FINCHARGEPRF	Finance Charge Preferences	/app/setup/finchargepref.nl
ADMI_FISCALPERIODS	Accountng Periods	/app/setup/period/generatefiscalperiods.nl
ADMI_GENERAL	General Preferences	/app/setup/general.nl
ADMI_HEADING	Heading	#
ADMI_IMAGERESIZE	Image Resizing	/app/site/setup/imageresizeadmin.nl
ADMI_IMPORTCSV	Import CSV Records	/app/setup/assistants/nsimport/importassistant.nl?new=T
ADMI_IMPORTCSV_LOG	View CSV Import Status	/app/setup/upload/csv/csvstatus.nl
ADMI_IMPORTOLB	Import Online Banking Data	/pages/setup/importolbform.jsp
ADMI_IMPORTQIF	Import Quicken&reg; QIF File	/pages/setup/importqifform.jsp
ADMI_IMPORT_COUPONCODE_STATUS	Import Coupon Codes Status	/app/setup/upload/csv/csvstatus.nl?importtype=COUPONCODE
ADMI_INTEGR_APP	Manage Integrations	/app/common/integration/integrapplist.nl

Task ID	Page Label in NetSuite	URL
ADMI_INVOICESETUP	Invoicing Preferences	/app/setup/invoicesetup.nl
ADMI_ISSUEBUILD	Product Builds	/app/crm/support/issuedb/issuebuilds.nl
ADMI_ISSUEEXTSTATUS	Issue External Statuses	/app/crm/support/issuedb/issueextstatus.nl
ADMI_ISSUEIMPORT	Import Issue Records	/app/setup/assistants/nsimport/importassistant.nl
ADMI_ISSUEMODULE	Product Modules	/app/crm/support/issuedb/issuemodule.nl
ADMI_ISSUEPRIORITY	Issue Priorities	/app/crm/support/issuedb/issuepriority.nl
ADMI_ISSUEPRODUCT	Products	/app/crm/support/issuedb/issueproducts.nl
ADMI_ISSUERELEASEUPDATE	Manage Released Issues	/app/crm/support/issuedb/issuereleaseupdate.nl
ADMI_ISSUEREPRODUCE	Issue Reproducibility	/app/crm/support/issuedb/issuereproduce.nl
ADMI_ISSUESETUP	Issue Preferences	/app/setup/issuesetup.nl
ADMI_ISSUESEVERITY	Issue Severities	/app/crm/support/issuedb/issueseverity.nl
ADMI_ISSUESOURCE	Issue Sources	/app/crm/support/issuedb/issuesource.nl
ADMI_ISSUESTATUS	Issue Statuses	/app/crm/support/issuedb/issuestatus.nl
ADMI_ISSUESTATUSFLOW	Manage Status Transitions	/app/crm/support/issuedb/issuestatusflow.nl
ADMI_ISSUETAGS	Issue Tags	/app/crm/support/issuedb/issuetags.nl
ADMI_ISSUETYPE	Issue Types	/app/crm/support/issuedb/issuetype.nl
ADMI_ISSUEUSERTYPE	Issue Roles	/app/crm/support/issuedb/issueusertypes.nl
ADMI_ISSUEVERSION	Product Versions	/app/crm/support/issuedb/issueversion.nl
ADMI_ITEMATTRGROUP	Field Set	/app/site/setup/fieldset.nl
ADMI_LEADCUSTOMFIELD MAPPING	Lead Conversion Mapping	/app/common/custom/custleadfieldmapping.nl
ADMI_LOGINAUDIT	View Login Audit Trail	/app/common/search/search.nl?searchtype=LoginAudit
ADMI_LOGINRESTRICT	View Login Restrictions	/app/setup/userloginrestrictions.nl
ADMI_MAINTENANCEDOMAIN	Site Maintenance Domain	/app/site/setup/maintainedomain.nl
ADMI_MAINTENANCEDOMAINADV	Site Maintenance Domain	/app/site/setup/maintainedomain.nl



Task ID	Page Label in NetSuite	URL
ADMI_MAINTENANCEDOMAINLIST	Site Maintenance Domains	/app/setup/maintenancedomains.nl
ADMI_MAINTENANCEDOMAINLISTADV	Site Maintenance Domains	/app/setup/maintenancedomains.nl
ADMI_MANAGEPAYROLL	Update Payroll Information	/app/payroll/managepayroll.nl
ADMI_MANAGE_PLUGINS	Manage Plug-ins	/app/common/scripting/manageplugins.nl
ADMI_NAMING	Rename Records/Transactions	/app/setup/naming.nl
ADMI_NETANSWERS	NetAnswers Knowledge Base	javascript:window.open("http://shopping.netsuite.com/netledger","NetAnswers","resizable,scrollbars=yes");void(0)
ADMI_NEXUSES	Nexus	/app/setup/nexuses.nl
ADMI_NOTIFICATIONS	Administrative Notifications	/app/setup/notifications.nl?id=0
ADMI_NUMBERING	Set Up Auto-Generated Numbers	/app/setup/numbering.nl
ADMI_OAUTH_TOKENS	Access Tokens	/app/setup/accesstokens.nl
ADMI_OPENIDSSO	OpenID Single Sign-on	/app/setup/openidsetup.nl
ADMI_OTHERSTUB	Other Section Task	#
ADMI_OUTLOOKINTEGRATION	Outlook Integration	/app/external/xml/outlook/outlookDownload.nl
ADMI_OUTLOOKINTEGRATION_V3	Outlook Integration 3.0	/app/external/xml/outlook/outlookv3download.nl
ADMI_PAYPAL	Set Up PayPal Processing	/app/common/otherlists/accountingotherlist.nl?paymentmethodtype=paypal&system=F&e=T
ADMI_PAYROLL	Set Up Payroll	/app/setup/payrollsetup.nl
ADMI_PAYROLLMAP	Map Payroll Items	/app/payroll/mappayrollitem.nl
ADMI_PAYROLLREP	Report Sections	/app/setup/payrollreportsections.nl
ADMI_PRINTING	Printing & Fax	/app/setup/printing.nl
ADMI_REDIRECT	Import Redirect	/app/setup/assistants/nsimport/simpleimport.nl?rectype=REDIRECT
ADMI_REDIRECTADV	Import Redirect	/app/setup/assistants/nsimport/simpleimport.nl?rectype=REDIRECT
ADMI_REDIRECTS	Web Site Redirects	/app/site/setup/redirects.nl
ADMI_REDIRECTSADV	Web Site Redirects	/app/site/setup/redirects.nl
ADMI_SAMLSSO	SAML Single Sign-on	/app/setup/samlsetup.nl
ADMI_SANDBOXACCOUNTS	Sandbox Accounts	/app/setup/sandboxaccounts.nl
ADMI_SAVEDASH	Publish Dashboard	/app/center/setup/savedashboard.nl



Task ID	Page Label in NetSuite	URL
ADMI_SAVEDIMPORTS	Saved CSV Imports	/app/setup/assistants/nsimport/savedimports.nl
ADMI_SCRIPTDEBUGGER	Script Debugger	/app/common/scripting/scriptdebugger.nl
ADMI_SETUPMANAGER	Setup Manager	/app/setup/mainsetup.nl
ADMI_SETUPURLS	Bulk Set URL Components	/app/site/setup/setupurlcomponents.nl
ADMI_SETUPURLSADV	Bulk Set URL Components	/app/site/setup/setupurlcomponents.nl
ADMI_SETUPURLSADV_LOG	Bulk Set URL Components Status	/app/external/xml/upload/uploadlog.nl?displayType=SITESETUP
ADMI_SETUPURLS_LOG	Bulk Set URL Components Status	/app/external/xml/upload/uploadlog.nl?displayType=SITESETUP
ADMI_SETUPYEARSTATUS	Set Up Year Status	/app/setup/period/generateperiodssstatus.nl
ADMI_SFASSETUP	Sales Preferences	/app/setup/sfasetup.nl
ADMI_SHIPPING	Set Up Shipping	/app/setup/shipping.nl
ADMI_SITEMAPGENERATOR	Sitemap Generator	/app/site/setup/sitemap/sitemapgenerator.nl
ADMI_SITEMAPGENERATORADV	Sitemap Generator	/app/site/setup/sitemap/sitemapgenerator.nl?sitetype=ADVANCED
ADMI_SITEMAP_MANAGER	Content Manager	/app/site/setup/sitemanager.nl
ADMI_SOFTDESCRIPTORS	Set Up Credit Card Soft Descriptors	/app/setup/softdescriptors.nl
ADMI_STATETAXIMPORT	Use State Sales Tax Tables	/app/setup/statetaximport.nl
ADMI_STOREADMIN	Set Up Web Site	/app/site/setup/siteadmin.nl?sitetype=STANDARD
ADMI_STOREADMINADV	Set Up Web Site	/app/site/setup/siteadmin.nl?sitetype=ADVANCED
ADMI_STOREASSISTANT	Web Site Assistant	/app/setup/assistants/sitesetup.nl
ADMI_STORELIST	Set Up Web Site	/app/site/setup/sitelist.nl?sitetype=STANDARD
ADMI_STORELISTADV	Set Up Web Site	/app/site/setup/sitelist.nl?sitetype=ADVANCED
ADMI_STOREPREVIEW	Preview Web Site	/app/site/setup/sitepreview.nl
ADMI_STOREPREVIEWADV	Preview Web Site	/app/site/setup/sitepreview.nl
ADMI_SUITESIGNON	SuiteSignOn	/app/setup/ssoapplist.nl
ADMI_SUPPORTSETUP	Support Preferences	/app/setup/supportsetup.nl
ADMI_SWAPPRICES	Swap Prices Between Price Levels	/app/common/item/swapprices.nl
ADMI_SYNCHRONIZATION SETUP	Synchronization	/app/external/xml/pumatech/syncDownload.nl



Task ID	Page Label in NetSuite	URL
ADMI_TAX	Set Up Taxes	/app/setup/taxpreferences.nl
ADMI_TAXACCTS	Tax Control Accounts	/app/setup/taxaccts.nl
ADMI_TAXTYPES	Tax Types	/app/setup/taxtypes.nl
ADMI_TEGATA	Tegata Accounts	/app/setup/tegataaccounts.nl
ADMI_TEXTCUST	Customize Web Site Text	/app/setup/customizetextbundles.nl
ADMI_TEXTCUSTADV	Customize Web Site Text	/app/setup/customizetextbundles.nl
ADMI_TEXTCUSTGROUP	Customize Text Group	/app/setup/customizetextbundle.nl
ADMI_TEXTCUSTGROUPAD V	Customize Text Group	/app/setup/customizetextbundle.nl
ADMI_TRANSETUP	Set Up Transactions	/app/setup/acctsetup.nl
ADMI_TRANSITEMTXT	Bulk Update Translation	/app/setup/translateitems.nl
ADMI_TWOFACTORDEVICE S	Two-Factor Authentication Tokens	/app/setup/twofactordevices.nl
ADMI_TWOFACTORPHONE RESET	Two-Factor Phone Number Reset	/app/setup/adminresetphone.nl
ADMI_TWOFACTORROLES	Two-Factor Authentication Roles	/app/setup/twofactorroles.nl
ADMI_UPDATEPRICES	Update Prices	/app/common/bulk/bulkop.nl? searchtype=Item&opcode=Pricing
ADMI_UPSELLSETUP	Upsell Preferences	/app/setup/upsellsetup.nl
ADMI_UPSWIZ	UPS Registration Wizard	/app/common/shipping/ups/ upsregistration.nl
ADMI_URLCOMPONENTS	URL Components for Facets	/app/site/setup/ faceturlcomponentmanager.nl
ADMI_WEBSERVICEPREFS	Web Services Preferences	/app/webservices/setup.nl
ADMI_WEBSERVICES_ STATUS	Web Services Process Status	/app/webservices/asyncstatus.nl
ADMI_WEBSERVICES_ USAGE_LOG	Web Services Usage Log	/app/webservices/syncstatus.nl
ADMI_XML_PAYTRUST_ APPROVE	Approve Online Bill Payments	/app/external/xml/paytrust/approve.nl
ADMI_XML_PAYTRUST_ SETUP	Set Up Online Bill Pay	/app/external/xml/paytrust/setup.nl
ADMI_XML_PAYTRUST_ STATUS	View Online Bill Pay Status	/app/external/xml/paytrust/status.nl
ADMI_YTDTAXLIBANDPYM TS	Set Up Year-to-Date Information	/app/payroll/ytdprocess.nl
EDIT_ACCOUNT	New Accounts	/app/accounting/account/account.nl
EDIT_ACCOUNTINGBOOK	New Accounting Book	/app/accounting/multibook/ accountingbook.nl



Task ID	Page Label in NetSuite	URL
EDIT_ACCOUNTINGOTHERLIST	New Accounting List Element	/app/common/otherlists/accountingotherlist.nl
EDIT_ACTIVITY	New Activity	/app/crm/calendar/activity.nl
EDIT_ADVANCEDORDERMANAGEMENTSETUP	Advanced Order Management	/app/setup/advancedordermanagement/advancedordermanagementsetup.nl
EDIT_ALLOCATION	Create Allocation Schedules	/app/accounting/transactions/allocation.nl
EDIT_ALLOCATIONBATCH	Create Allocation Batches	/app/accounting/transactions/allocationbatch.nl
EDIT_AMENDW4	Form W-4	/app/common/entity/amendw4.nl
EDIT_AMORTIZATIONSCHEDULE	New Amortization Template	/app/accounting/otherlists/revrecschedule.nl?type=Amortization
EDIT_BILLINGACCOUNT	Create Billing Accounts	/app/accounting/otherlists/billingaccount.nl
EDIT_BILLINGCLASS	New Billing Class	/app/accounting/otherlists/billingclass.nl
EDIT_BILLINGRULE	New Billing Rule	/app/accounting/transactions/billing/billingrule.nl
EDIT_BILLINGSCHEDULE	New Billing Schedule	/app/accounting/otherlists/billingschedule.nl
EDIT_BILLING_GROUPS	Billing Groups	/app/crm/common/crmgroup.nl?billinggroupsonly=T&groupType_filter=CustJob
EDIT_BILLOFDISTRIBUTION	New Bill Of Distribution	/app/accounting/inventory/distributionplanning/billofdistribution.nl
EDIT_BILLRUNSCHEDULE	Schedule Billing Operations	/app/accounting/transactions/billingworkcenter/billrunschedule.nl
EDIT_BILLRUNSCHEDULES	New Billing Operation Schedule	/app/accounting/transactions/billingworkcenter/billrunschedule.nl
EDIT_BINNUMBERRECORD	New Bin	/app/accounting/transactions/inventory/binnumberrecord.nl
EDIT_BULKOP	Edit Mass Update	/app/common/bulk/bulkop.nl
EDIT_BUNDLE	Create Bundle	/app/setup/assistants/bundlebuilder.nl?new=T
EDIT_BUNDLEAUDITTRAIL	Bundle Audit Trail	/app/bundler/bundleaudittrail.nl
EDIT_CALENDARPREFERENCE	Calendar Preference	/app/crm/calendar/calendarpreference.nl
EDIT_CALL	New Phone Call	/app/crm/calendar/call.nl
EDIT_CAMPAIGN	New Marketing Campaign	/app/crm/marketing/campaign.nl
EDIT_CAMPAIGNAUDIENCE	New Campaign Audience	/app/crm/marketing/campaignaudience.nl

Task ID	Page Label in NetSuite	URL
EDIT_CAMPAIGNBULK	Create Keyword Campaigns	/app/crm/marketing/campaign.nl?bulk=T
EDIT_CAMPAIGNBULKIMPORT	Import Keywords	/app/setup/assistants/nsimport/simpleimport.nl?rectype=CAMPAIGNKEYWORD
EDIT_CAMPAIGNCATEGORY	New Campaign Category	/app/crm/marketing/campaigncategory.nl
EDIT_CAMPAIGNCHANNEL	New Campaign Channel	/app/crm/marketing/campaignchannel.nl
EDIT_CAMPAIGNEMAIL	New Campaign Email Address	/app/crm/marketing/campaignemail.nl
EDIT_CAMPAIGNFAMILY	New Campaign Family	/app/crm/marketing/campaignfamily.nl
EDIT_CAMPAIGNOFFER	New Campaign Offer	/app/crm/marketing/campaignoffer.nl
EDIT_CAMPAIGNSEARCHENGINE	New Campaign Search Engine	/app/crm/marketing/campaignsearchengine.nl
EDIT_CAMPAIGNSUBSCRIPTION	New Campaign Subscription	/app/crm/marketing/campaignsubscription.nl
EDIT_CAMPAIGNVERTICAL	New Campaign Vertical	/app/crm/marketing/campaignvertical.nl
EDIT_CASEFIELDRULE	New Case Rule	/app/crm/support/casefieldrule.nl
EDIT_CASEFORM	New Online Case Forms	/app/crm/support/caseform.nl
EDIT_CASEISSUE	New Case Issue	/app/crm/support/caseissue.nl
EDIT_CASEORIGIN	New Case Origin Type	/app/crm/support/caseorigin.nl
EDIT_CASEPRIORITY	New Case Priority	/app/crm/support/casepriority.nl
EDIT_CASEPROFILE	New Case Profile	/app/crm/support/profiles/caseprofile.nl
EDIT_CASESTATUS	New Case Status	/app/crm/support/casestatus.nl
EDIT_CASETERRITORY	New Case Territory	/app/crm/support/supportterritory.nl
EDIT_CASETYPE	New Case Type	/app/crm/support/casetype.nl
EDIT_CHARGE	Create Charges	/app/accounting/transactions/billing/charge.nl
EDIT_CHARGERULE	New Charge Rule	/app/accounting/transactions/billing/chargerule.nl
EDIT_CLASS	New Class	/app/common/otherlists/classtype.nl
EDIT_CLASSSEGMENTMAPPING	New Class Mapping	/app/accounting/account/classsegmentmapping.nl
EDIT_COLORTHEME	New Color Theme	/app/setup/look/colortheme.nl
EDIT_COMMERCECATALOG	Manage Commerce Category Catalog	/app/platform/services/commercecategory/commercecatalog.nl
EDIT_COMMERCECATEGORY	Manage Commerce Category	/app/platform/services/commercecategory/commercecategory.nl



Task ID	Page Label in NetSuite	URL
EDIT_COMMISONSCHEDULE	New Employee Schedule	/app/crm/sales/commissions/commissionschedule.nl
EDIT_COMPANY	New Company	/app/common/entity/company.nl
EDIT_COMPETITOR	Competitor	/app/crm/sales/competitor.nl
EDIT_CONTACT	New Contacts	/app/common/entity/contact.nl
EDIT_CRMGROUP	New Groups	/app/crm/common/crmgroup.nl
EDIT_CRMMESSAGE	New Email	/app/crm/common/crmmessage.nl
EDIT_CRMOTHERLIST	New CRM List Element	/app/common/otherlists/crmotherlist.nl
EDIT_CRMTEMPLATE	New Marketing Templates	/app/crm/common/merge/marketingtemplate.nl
EDIT_CURRENCY	New Currencies	/app/common/multicurrency/currency.nl
EDIT_CURRENCYRATE	New Currency Exchange Rate	/app/common/multicurrency/currencyrate.nl
EDIT_CUSTADDRESSENTRYFORM	Address Form	/app/common/custom/custaddressentryform.nl
EDIT_CUSTADDRESSFORM	Address Form	/app/common/custom/custaddressform.nl?e=T
EDIT_CUSTBODYFIELD	New Transaction Body Fields	/app/common/custom/bodycustfield.nl
EDIT_CUSTCATEGORY	New Center Category	/app/common/custom/custcategory.nl
EDIT_CUSTCENTER	New Center	/app/common/custom/custcenter.nl
EDIT_CUSTCOLUMNFIELD	New Transaction Column Fields	/app/common/custom/columncustfield.nl
EDIT_CUSTEMAILLAYOUT	New Transaction Form HTML Layouts	/app/common/custom/custemaillayout.nl
EDIT_CUSTENTITYFIELD	New Entity Fields	/app/common/custom/entitycustfield.nl
EDIT_CUSTENTRYFORM	New Entry Forms	/app/common/custom/custentryform.nl
EDIT_CUSTEVENTFIELD	New CRM Fields	/app/common/custom/eventcustfield.nl
EDIT_CUSTFORM	New Transaction Forms	/app/common/custom/custform.nl
EDIT_CUSTITEMFIELD	New Item Fields	/app/common/custom/itemcustfield.nl
EDIT_CUSTITEMNUMBERFIELD	New Item Number Field	/app/common/custom/itemnumbercustfield.nl
EDIT_CUSTJOB	New Customers	/app/common/entity/custjob.nl
EDIT_CUSTLAYOUT	New Transaction Form PDF Layouts	/app/common/custom/custlayout.nl
EDIT_CUSTLIST	New Lists	/app/common/custom/custlist.nl
EDIT_CUSTOMERFIELDRULE	New Set Up Sales Rules	/app/crm/sales/customerfieldrule.nl
EDIT_CUSTOMERFORM	New Online Customer Forms	/app/crm/sales/leadform.nl

Task ID	Page Label in NetSuite	URL
EDIT_CUSTOMERSTATUS	New Customer Statuses	/app/crm/sales/customerstatus.nl
EDIT_CUSTOMSEGMENT	Edit Custom Segment	/app/common/custom/segments/segment.nl
EDIT_CUSTOTHERFIELD	New Other Field	/app/common/custom/othercustfield.nl
EDIT_CUSTPROFILE	Customer Profile	/app/common/entity/custprofile.nl?category=billing&sc=4
EDIT_CUSTRECORD	New Record Types	/app/common/custom/custrecord.nl
EDIT_CUSTRECORDFIELD	Custom Record Field	/app/common/custom/custreccustfield.nl
EDIT_CUSTRECORDFORM	New Custom Record Form	/app/common/custom/custrecordform.nl
EDIT_CUSTSCRIPTFIELD	New Script Fields	/app/common/custom/scriptcustfield.nl
EDIT_CUSTSECTION	New Center Tab	/app/common/custom/custsection.nl
EDIT_CUSTTAB	New Subtab	/app/common/custom/subtab.nl
EDIT_CUSTTASKS	Center Links	/app/common/custom/custtasks.nl
EDIT_CUSTTRANSACTION	New Transaction Type	/app/common/custom/customtransaction.nl
EDIT_CUSTWFSTATEFIELD	New Workflow State Fields	/app/common/custom/wfstatecustfield.nl
EDIT_CUSTWORKFLOWFIELD	New Workflow Fields	/app/common/custom/workflowcustfield.nl
EDIT_CUST_	Custom Record Entry	/app/common/custom/custrecordentry.nl
EDIT_DEPARTMENT	New Departments	/app/common/otherlists/departmenttype.nl
EDIT_DEPLOYMENTAUDITTRAIL	Deployment Audit Trail	/app/suiteapp/devframework/deploymentaudittrail.nl
EDIT_DEPTSEGMENTMAPPING	New Department Mapping	/app/accounting/account/deptsegmentmapping.nl
EDIT_DISTRIBUTIONNETWORK	New Distribution Network	/app/accounting/inventory/distributionplanning/distributionnetwork.nl
EDIT_DRIVERSLICENSE	New Driver's License	/app/hcm/hris/governmentid/driverslicense.nl
EDIT_EDITPROFILE	Employee Profile	/app/common/entity/editprofile.nl
EDIT_EMAILTEMPLATE	New Email Templates	/app/crm/common/merge/emailtemplate.nl
EDIT_EMPLCATEGORY	New Employee Directory	/app/site/setup/emplcategory.nl
EDIT_EMPLOYEE	New Employees	/app/common/entity/employee.nl
EDIT_EMPLOYEESFA	New Assign Support Reps	/app/common/entity/employeesfa.nl

Task ID	Page Label in NetSuite	URL
EDIT_EMPOTHERLIST	New Employee Related List Element	/app/common/otherlists/empotherlist.nl
EDIT_ENTITYACCOUNTAPPING	New Entity Account Mapping	/app/accounting/account/entityaccountmapping.nl
EDIT_ESCALATIONRULE	New Escalation Rules	/app/crm/support/escalationfieldrule.nl
EDIT_ESCALATIONTERRITORY	New Manage Escalation Assignment	/app/crm/support/escalationterritory.nl
EDIT_EVENT	New Event	/app/crm/calendar/event.nl
EDIT_EXPCATEGORY	New Expense Categories	/app/accounting/otherlists/expcategory.nl
EDIT_FAIRVALUEDIMENSION	Fair Value Dimension	/app/accounting/revrec/fairvaluedimension.nl
EDIT_FAIRVALUEFORMULA	New Fair Value Formula	/app/accounting/revrec/fairvalueformula.nl
EDIT_FAIRVALUEPRICE	New Fair Value Price	/app/accounting/revrec/fairvalueprice.nl
EDIT_FAXMESSAGE	New Fax Message	/app/crm/common/merge/faxmessage.nl
EDIT_FAXTEMPLATE	New Fax Templates	/app/crm/common/merge/faxtemplate.nl
EDIT_FISCALCALENDAR	New Fiscal Calendar	/app/setup/period/fiscalcalendar.nl
EDIT_FISCALPERIOD	Setup Fiscal Period	/app/setup/period/fiscalperiod.nl
EDIT_GENERICRESOURCE	New Generic Resource	/app/accounting/project/genericresource.nl
EDIT_GLOBALACCOUNTAPPING	New Global Account Mapping	/app/accounting/account/globalaccountmapping.nl
EDIT_GOVERNMENTISSUEDIDTYPE	New Government-Issued ID Type	/app/hcm/hris/governmentid/governmentissuedidtype.nl
EDIT_HCMJOB	New Job	/app/hcm/hris/job.nl
EDIT_IC_ALLOCATION	Create Intercompany Allocation Schedules	/app/accounting/transactions/intercompanyallocation.nl
EDIT_IMPORT_COUPONCODE	Import Coupon Codes	/app/setup/assistants/nsimport/simpleimport.nl?rectype=COUPONCODE
EDIT_INFOITEM	New Information Items	/app/site/setup/infoitem.nl?Information_TYPE=TEXT
EDIT_INFOITEMFORM	New Publish Forms	/app/site/setup/infoitem.nl?Information_TYPE=FORM
EDIT_INSTALLBUNDLE	Search & Install Bundles	/app/bundler/installbundle.nl
EDIT_INTEGR_APP	Integration	/app/common/integration/integrapp.nl
EDIT_INVCOSTTEMPLATE	New Inventory Cost Template	/app/accounting/inventory/standard/invcosttemplate.nl
EDIT_ISSUE	New Issue	/app/crm/support/issuedb/issue.nl



Task ID	Page Label in NetSuite	URL
EDIT_ISSUEPRODUCT	New Product	/app/crm/support/issuedb/issueproduct.nl
EDIT_ISSUETAG	New Issue Tag	/app/crm/support/issuedb/issuetag.nl
EDIT_ISSUEUSERTYPE	New Issue Role	/app/crm/support/issuedb/issueusertype.nl
EDIT_ITEM	New Items	/app/common/item/item.nl
EDIT_ITEMACCOUNTMAPPING	New Item Account Mapping	/app/accounting/account/itemaccountmapping.nl
EDIT_ITEMDEMANDPLAN	New Item Demand Plan	/app/accounting/inventory/demandplanning/itemdemandplan.nl
EDIT_ITEMOPTION	New Transaction Item Options	/app/common/custom/itemoption.nl
EDIT_ITEMREVENUECATEGORY	New Item Revenue Category	/app/accounting/revrec/itemrevenuecategory.nl
EDIT_ITEMSUPPLYPLAN	New Item Supply Plan	/app/accounting/inventory/demandplanning/itemsupplyplan.nl
EDIT_ITEM_REVISION	New Item Revision	/app/common/item/itemrevision.nl
EDIT_JOB	New Job	/app/accounting/project/project.nl
EDIT_JOBREQUISITION	New Job Requisition	/app/hcm/hris/jobrequisition.nl
EDIT_KBCATEGORY	New Knowledge Base	/app/site/setup/kbcategory.nl
EDIT_KPIREPORT	New KPI Scorecard	/app/center/enhanced/kpireportsetup.nl
EDIT_KUDOS	New Kudos	/app/hcm/perfmgmt/kudos.nl
EDIT_LEAD	New Leads	/app/common/entity/custjob.nl?stage=lead
EDIT_LOCASSIGNCONF	Configurations	/app/setup/locationassignment/configuration/locationassignmentconfiguration.nl
EDIT_LOCASSIGNRULE	Rules Edit	/app/setup/locationassignment/configuration/rule/locationassignmentrule.nl
EDIT_LOCASSIGNRULES_AUDIT_TRAIL	Rules Audit Trail	/app/common/search/search.nl?searchtype=LocAssignRule&audittrail=T
EDIT_LOCATION	New Locations	/app/common/otherlists/locationtype.nl
EDIT_LOCATIONCOSTINGGROUP	New Location Costing Group	/app/accounting/inventory/costing/locationcostinggroup.nl
EDIT_LOCSEGMENTMAPPING	New Location Mapping	/app/accounting/account/locsegmentmapping.nl
EDIT_MAILMERGE	Bulk Merge	/app/crm/common/merge/mailmerge.nl
EDIT_MAILMESSAGE	New Word Message	/app/crm/common/merge/mailmessage.nl
EDIT_MAILTEMPLATE	New Letter Templates	/app/crm/common/merge/mailtemplate.nl

Task ID	Page Label in NetSuite	URL
EDIT_MEDIAITEM	New Media Item	/app/common/media/mediaitem.nl
EDIT_MEDIAITEMFOLDER	New File Cabinet	/app/common/media/mediaitemfolder.nl
EDIT_MEMDOC	Enter Memorized Transactions	/app/accounting/transactions/bulkmemdoc.nl
EDIT_MFGCOSTTEMPLATE	New Manufacturing Cost Template	/app/accounting/manufacturing/mfgcosttemplate.nl
EDIT_MFGROUTING	New Manufacturing Routing	/app/accounting/manufacturing/mfgrouting.nl
EDIT_NEXUS	Nexus	/app/setup/nexus.nl
EDIT_NOTIFICATION	New Notification	/app/email/alerts/notification.nl
EDIT_OAUTHTOKEN	New Access Token	/app/setup/accesstoken.nl
EDIT_OPENIDSSO	OpenID Single Sign-on	/app/setup/openidsetup.nl
EDIT_ORGANIZATIONVALUE	New Organization Value	/app/hcm/perfmgmt/organizationvalue.nl
EDIT_OTHERGOVERNMENTISSUEDID	New Other Government-Issued ID	/app/hcm/hris/governmentid/othergovernmentissuedid.nl
EDIT_OTHERNAME	New Other Names	/app/common/entity/othername.nl
EDIT_PARTNER	New Partners	/app/common/entity/partner.nl
EDIT_PARTNERCOMMISSIONSCHED	New Partner Schedule	/app/crm/sales/commissions/partnercommissionschedule.nl
EDIT_PARTNERPLANASSIGN	New Partner Plan	/app/crm/sales/commissions/planassignpartner.nl
EDIT_PASSPORT	New Passport	/app/hcm/hris/governmentid/passport.nl
EDIT_PAYMETHODS	Create Payment Methods	/app/common/otherlists/accountingotherlist.nl?tname=paymeth&longitemtype=Payment+Method
EDIT_PAYPALACCOUNT	PayPal Accounts	/app/external/paypal/paypalaccount.nl
EDIT_PAYROLLBATCH2	New Payroll Batch	/app/payroll/payrollbatch2.nl
EDIT_PAYROLLITEM	New Payroll Items	/app/common/item/payrollitem.nl
EDIT_PAYTERMS	Create Payment Terms	/app/common/otherlists/accountingotherlist.nl?tname=terms&longitemtype=Term
EDIT_PDFMESSAGE	New PDF Message	/app/crm/common/merge/pdfmessage.nl
EDIT_PDFTEMPLATE	New PDF Templates	/app/crm/common/merge/pdftemplate.nl
EDIT_PERIOD	New Manage Accounting Periods	/app/setup/period/fiscalperiod.nl

Task ID	Page Label in NetSuite	URL
EDIT_PLANASSIGN	New Employee Plan	/app/crm/sales/commissions/planassign.nl
EDIT_PLANNEDSTANDARD COST	New Planned Standard Cost	/app/accounting/inventory/plannedstandardcost.nl
EDIT_PLUGIN	New Plug-in	/app/common/scripting/uploadPluginImpl.nl
EDIT_PLUGINTYPE	New Custom Plug-in Type	/app/common/scripting/createPluginType.nl
EDIT_POSITION	New Position	/app/hcm/hris/position.nl
EDIT_PRESCATEGORY	New Categories	/app/site/setup/prescategory.nl
EDIT_PROCESSHISTTXN	Historical Transaction Processing	/app/accounting/multibook/htp/submission.nl
EDIT_PROCESSHISTTXNLOG	Historical Transaction Processing Log	/app/accounting/multibook/htp/log.nl
EDIT_PROJECTEXPENSETYPE	New Project Expense Type	/app/accounting/otherlists/projectexpensetype.nl
EDIT_PROJECTTASK	New Job Tasks	/app/accounting/project/projecttask.nl
EDIT_PROJECTTEMPLATE	New Project Template	/app/accounting/project/projecttemplate.nl
EDIT_PROSPECT	New Prospects	/app/common/entity/custjob.nl?stage=prospect
EDIT_PUBLISHER	Register Publisher	/app/suiteapp/devframework/publisherregistry.nl
EDIT_PUBLISHERAPP	New Application	/app/suiteapp/devframework/appregistry.nl
EDIT_QUANTITYPRICINGSCHEDULE	New Quantity Pricing Schedule	/app/accounting/otherlists/quantitypricingschedule.nl
EDIT_RATEPLAN	New Rate Plan	/app/accounting/otherlists/rateplan.nl
EDIT_REDIRECT	New Redirect	/app/site/setup/redirect.nl
EDIT_REDIRECTADV	New Redirect	/app/site/setup/redirect.nl
EDIT_REFERRALCODE	New Promotion	/app/crm/sales/referralcode.nl
EDIT_REGION	New Region	/app/setup/region/region.nl
EDIT_REGISTERPUBLISHER	Register Publisher	/app/suiteapp/devframework/publisherregistry.nl
EDIT RELATEDITEM	New Related Items Category	/app/site/setup/relateditem.nl
EDIT RELATEDITEMADV	New Related Items Category	/app/site/setup/relateditem.nl
EDIT_REPORT	Edit Report	/app/reporting/reporteditor.nl
EDIT_RESOLVECONFLICTS	Resolve Conflicts	/app/common/entity/conflictresolution.nl
EDIT_RESOURCE	Resource	/app/crm/calendar/resource.nl



Task ID	Page Label in NetSuite	URL
EDIT_REVENUEALLOCATIO NGROUP	New Revenue Allocation Group	/app/common/otherlists/ accountingotherlist.nl?tname= revenueallocationgroup
EDIT_REVENUEELEMENT	New Revenue Element	/app/accounting/revrec/ revenueelement.nl
EDIT_REVENUEPLAN	New Revenue Recognition Plan	/app/accounting/revrec/revenueplan.nl
EDIT_REVENUERECOGNITI ONRULE	New Revenue Recognition Rule	/app/accounting/revrec/ revenuerecognitionrule.nl
EDIT_REVRECFIELDMAPPIN G	Revenue Recognition Field Mapping	/app/accounting/revrec/ revrecfieldmapping.nl
EDIT_REVRECSCHEDULE	New Revenue Recognition Template	/app/accounting/otherlists/ revrecschedule.nl
EDIT_ROLE	New Role	/app/setup/role.nl
EDIT_RSRCALLOCATION	New Rsrc Allocation	/app/accounting/project/allocation.nl
EDIT_RSSFEED	New RSS Feed	/app/site/hosting/rssfeed.nl
EDIT_SALESCAMPAIGN	New Sales Campaign	/app/crm/marketing/salescampaign.nl
EDIT_SALESTEAM	New Sales Teams	/app/crm/common/crmgroup.nl? groupType=SalesTeam
EDIT_SALESTERRITORY	New Manage Sales Territories	/app/crm/sales/salesterritory.nl
EDIT_SAVEDSEARCH	New Saved Search	/app/common/search/search.nl?cu=T& e=F
EDIT_SCRIPT	New Script	/app/common/scripting/ uploadScriptFile.nl
EDIT_SCRIPTEDRECORD	Scripted Record	/app/common/scripting/scriptedrecord. nl
EDIT_SEARCH	New Search	/app/common/search/search.nl
EDIT_SHIPITEM	New Shipping Items	/app/common/item/shipitem.nl
EDIT_SHIPPARTREGISTRAT ION	New Shipping Partner Registration	/app/common/shipping/openapi/ registration.nl
EDIT_SITEEMAILTEMPLATE	New Web Store Email Template	/app/site/setup/siteemailtemplate.nl
EDIT_SITEITEMTEMPLAT	Item Template	/app/site/setup/siteitemtemplate.nl
EDIT_SITEMEDIA	Site Media	/app/site/media/sitemedia.nl
EDIT_SITETAG	Web Site Tag	/app/site/setup/sitetag.nl
EDIT_SITETHHEME	Web Site Theme	/app/site/setup/sitetheme.nl
EDIT SOLUTION	New Solutions	/app/crm/support/kb/solution.nl
EDIT_SSATEGORY	New Publish Saved Search	/app/site/setup/sscategory.nl
EDIT_STANDARDCOSTVER SION	New Standard Cost Version	/app/accounting/inventory/ standardcostversion.nl
EDIT_STATE	New State/Province/County	/app/setup/state.nl

Task ID	Page Label in NetSuite	URL
EDIT_STOREITEMLISTLAYOUT	New Layouts	/app/site/setup/storeitemlistlayout.nl
EDIT_STORETAB	New Tabs	/app/site/setup/storetab.nl
EDIT_SUBSCRIPTION	Create Subscriptions	/app/accounting/subscription/subscription.nl
EDIT_SUBSCRIPTIONCHANGEORDER	Manage Subscriptions	/app/accounting/subscription/subscriptionchangeorder.nl
EDIT_SUBSCRIPTIONLINE	New Subscription Line	/app/accounting/subscription/subscriptionline.nl
EDIT_SUBSCRIPTIONPLAN	New Subscription Plan	/app/common/item/item.nl?itemtype=SubscriPlan
EDIT_SUBSIDIARY	New Subsidiaries	/app/common/otherlists/subsidiarytype.nl
EDIT_SUITEAPPLIST	Installed SuiteApp List	/app/suiteapp/devframework/appinstalllist.nl
EDIT_SUITESIGNON	SuiteSignOn	/app/setup/ssoapp.nl
EDIT_SUPPORTCASE	New Cases	/app/crm/support/supportcase.nl
EDIT_SYSALERT	System Alert	/app/common/otherlists/systemalert.nl
EDIT_TASK	New Tasks	/app/crm/calendar/task.nl
EDIT_TAXACCT	Tax Control Account	/app/setup/taxacct.nl
EDIT_TAXGROUP	New Tax Groups	/app/common/item/taxgroup.nl
EDIT_TAXITEM	New Tax Codes	/app/common/item/taxitem.nl
EDIT_TAXPERIOD	New Manage Tax Periods	/app/setup/period/taxperiod.nl
EDIT_TAXSCHEDULE	New Tax Schedule	/app/common/item/taxschedule.nl
EDIT_TAXTYPE	Tax Type	/app/setup/taxtype.nl
EDIT_TEMPLATECATEGORY	New Template Category	/app/crm/common/merge/templatecategory.nl
EDIT_TERMINATIONREASON	New Termination Reason	/app/hcm/hris/terminationreason.nl
EDIT_TIMEOFFCHANGE	New Time-Off Change	/app/hcm/hris/timeoffmanagement/timeoffchange.nl
EDIT_TIMEOFFPLAN	New Time-Off Plan	/app/hcm/hris/timeoffmanagement/timeoffplan.nl
EDIT_TIMEOFFREQUEST	New Time-Off Request	/app/hcm/hris/timeoffmanagement/timeoffrequest.nl
EDIT_TIMEOFFRULE	New Time-Off Rule	/app/hcm/hris/timeoffmanagement/timeoffrule.nl
EDIT_TIMEOFFTYPE	New Time-Off Type	/app/hcm/hris/timeoffmanagement/timeofftype.nl
EDIT_TIMESHEET	Enter Time	/app/accounting/transactions/timesheet.nl

Task ID	Page Label in NetSuite	URL
EDIT_TOPIC	New Topics	/app/crm/support/kb/topic.nl
EDIT_TRANSACTIONLIST	Paycheck History	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Paycheck
EDIT_TRAN_ADJJOURNAL	Make Currency Adjustment Journal Entries	/app/accounting/transactions/journal.nl?adjustmentjournal=T
EDIT_TRAN_BINTRNFR	Bin Transfer	/app/accounting/transactions/bintrnfr.nl
EDIT_TRAN_BINWKSHT	Bin Putaway Worksheet	/app/accounting/transactions/binwksht.nl
EDIT_TRAN_BLANKORD	Enter Blanket Purchase Order	/app/accounting/transactions/blankord.nl
EDIT_TRAN_BOOKICJOURNAL	Make Book Specific Intercompany Journal Entries	/app/accounting/transactions/journal.nl?bookje=T&icj=T
EDIT_TRAN_BOOKICJOURNALIMPORT	Import Book Specific Intercompany Journal Entries	/app/setup/assistants/nsimport/simpleimport.nl?rectype=INTERCOMPANYJOURNALENTRY
EDIT_TRAN_BOOKJOURNAL	Make Book Specific Journal Entries	/app/accounting/transactions/journal.nl?bookje=T
EDIT_TRAN_BOOKJOURNALIMPORT	Import Book Specific Journal Entries	/app/setup/assistants/nsimport/simpleimport.nl?rectype=JOURNALENTRY
EDIT_TRAN_BUILD	Build Assemblies	/app/accounting/transactions/build.nl
EDIT_TRAN_CARDCHRG	Use Credit Card	/app/accounting/transactions/cardchrg.nl
EDIT_TRAN_CASHRFND	Refund Cash Sales	/app/accounting/transactions/cashrfnd.nl
EDIT_TRAN_CASHSALE	Enter Cash Sales	/app/accounting/transactions/cashsale.nl
EDIT_TRAN_CHECK	Write Checks	/app/accounting/transactions/check.nl
EDIT_TRAN_COMMISSN	Individual Employee Commission	/app/accounting/transactions/commissn.nl
EDIT_TRAN_CUSTCHRG	Create Statement Charges	/app/accounting/transactions/custchrg.nl
EDIT_TRAN_CUSTCRED	Issue Credit Memos	/app/accounting/transactions/custcred.nl
EDIT_TRAN_CUSTDEP	Record Customer Deposits	/app/accounting/transactions/custdep.nl
EDIT_TRAN_CUSTINVC	Create Invoices	/app/accounting/transactions/custinvc.nl
EDIT_TRAN_CUSTPYMT	Accept Customer Payments	/app/accounting/transactions/custpymt.nl
EDIT_TRAN_CUSTRFND	Issue Customer Refund	/app/accounting/transactions/custrfnd.nl

Task ID	Page Label in NetSuite	URL
EDIT_TRAN_DEPAPPL	Apply Customer Deposits	/app/accounting/transactions/depappl.nl
EDIT_TRAN_DEPOSIT	Make Deposits	/app/accounting/transactions/deposit.nl
EDIT_TRAN_ESTIMATE	Prepare Estimates	/app/accounting/transactions/estimate.nl
EDIT_TRAN_EXPREPT	Enter Expense Reports	/app/accounting/transactions/exprept.nl
EDIT_TRAN_FFTREQ	Edit	/app/accounting/transactions/fftreq.nl
EDIT_TRAN_FXREVAL	Revalue Open Currency Balances	/app/accounting/transactions/fxrevalsetup.nl
EDIT_TRAN_ICJOURNAL	Make Intercompany Journal Entries	/app/accounting/transactions/journal.nl?icj=T
EDIT_TRAN_ICJOURNALIMPORT	Import Intercompany Journal Entries	/app/setup/assistants/nsimport/simpleimport.nl?rectype=INTERCOMPANYJOURNALENTRY
EDIT_TRANICTRNFRORD	Enter Intercompany Transfer Orders	/app/accounting/transactions/trnfrord.nl?icto=T
EDIT_TRAN_INVADJST	Adjust Inventory	/app/accounting/transactions/invadjst.nl
EDIT_TRAN_INVCOUNT	Enter Inventory Count	/app/accounting/transactions/invcount.nl
EDIT_TRAN_INVDISTR	Distribute Inventory	/app/accounting/transactions/invdistr.nl
EDIT_TRAN_INVREVAL	Revalue Inventory Cost	/app/accounting/transactions/invreval.nl
EDIT_TRAN_INVTRNFR	Transfer Inventory	/app/accounting/transactions/invtrnfr.nl
EDIT_TRAN_INVWKSHT	Adjust Inventory Worksheet	/app/accounting/transactions/invwksht.nl
EDIT_TRAN_INVWKSHTIMPORT	Import Inventory Worksheets	/app/setup/assistants/nsimport/simpleimport.nl?rectype=INVENTORYWORKSHEET
EDIT_TRAN_ITEMRCPT	New Item Receipt	/app/accounting/transactions/itemrcpt.nl
EDIT_TRAN_ITEMSHIP	New Item Fulfillment	/app/accounting/transactions/itemship.nl
EDIT_TRAN_JOURNAL	Make Journal Entries	/app/accounting/transactions/journal.nl
EDIT_TRAN_JOURNALIMPORT	Import Journal Entries	/app/setup/assistants/nsimport/simpleimport.nl?rectype=JOURNALENTRY
EDIT_TRAN_LIAADJST	Liability Adjustment	/app/accounting/transactions/liaadjst.nl
EDIT_TRAN_LIABPYMT	Pay Payroll Liabilities	/app/accounting/transactions/liabpymt.nl
EDIT_TRAN_OPRTNTY	Create Opportunities	/app/accounting/transactions/opprndty.nl
EDIT_TRAN_ORDERPURCHREQ	Order Requisitions	/app/accounting/transactions/orderpurchreqs.nl

Task ID	Page Label in NetSuite	URL
EDIT_TRAN_PARTNERCOMMISSN	Individual Partner Commission	/app/accounting/transactions/partnercommissn.nl
EDIT_TRAN_PAYCHECK	Create Individual Paycheck	/app/accounting/transactions/paycheck.nl
EDIT_TRAN_PAYCHECK2	Create Individual Paycheck	/app/accounting/transactions/paycheck2.nl
EDIT_TRAN_PCHKJRNL	Paycheck Journal	/app/accounting/transactions/pchkjrnл.nl
EDIT_TRAN_PURCHCON	Enter Purchase Contracts	/app/accounting/transactions/purchcon.nl
EDIT_TRAN_PURCHORD	Enter Purchase Orders	/app/accounting/transactions/purchord.nl
EDIT_TRAN_PURCHORD_REQ	Enter Purchase Requests	/app/accounting/transactions/purchord.nl
EDIT_TRAN_PURCHREQ	Enter Requisitions	/app/accounting/transactions/purchreq.nl
EDIT_TRAN_REPLENISHINVENTORY	Replenish Location By Transfer Order	/app/accounting/transactions/inventory/bulktransferorder/replenishinventory.nl
EDIT_TRAN_REPLENISHLOC	Replenish Location By Inventory Transfer	/app/accounting/transactions/replenishloc.nl
EDIT_TRAN_REVARRNG	Enter Revenue Arrangements	/app/accounting/transactions/revarrng.nl
EDIT_TRAN_REVCOMM	RevComm	/app/accounting/transactions/revcomm.nl
EDIT_TRAN_REVCOMRV	RevComRv	/app/accounting/transactions/revcomrv.nl
EDIT_TRAN_REVCONTR	RevContr	/app/accounting/transactions/revcontr.nl
EDIT_TRAN_RFQ	Enter Requests For Quote	/app/accounting/transactions/rfq.nl
EDIT_TRAN_RTNAUTH	Issue Return Authorizations	/app/accounting/transactions/rtnauth.nl
EDIT_TRAN_SALESORD	Enter Sales Orders	/app/accounting/transactions/salesord.nl
EDIT_TRAN_STATJOURNAL	Make Statistical Journal Entries	/app/accounting/transactions/statisticaljournal.nl
EDIT_TRAN_STATJOURNALIMPORT	Import Statistical Journal Entries	/app/setup/assistants/nsimport/simpleimport.nl?rectype=STATISTICALJOURNALENTRY
EDIT_TRAN_STATSCHEDULE	Create Statistical Schedule	/app/accounting/transactions/statistical/statisticschedule.nl
EDIT_TRAN_STPICKUP	Edit	/app/accounting/transactions/stpickup.nl
EDIT_TRAN_TAXLIAB	Write GST Liability	/app/accounting/transactions/vatliab.nl
EDIT_TRAN_TAXLIAB2	Write Tax Liability	/app/accounting/transactions/vatliab.nl

Task ID	Page Label in NetSuite	URL
EDIT_TRAN_TAXPYMT	Pay Sales Tax	/app/accounting/transactions/taxpymt.nl
EDIT_TRAN_TAXPYMT2	Pay Sales Tax	/app/accounting/transactions/taxpymt.nl
EDIT_TRAN_TAXPYMTCA	Pay PST	/app/accounting/transactions/taxpymt.nl
EDIT_TRAN_TEGCOLLECT	Collect Tegata	/app/accounting/transactions/tegcollect.nl
EDIT_TRAN_TEGPAY	Pay Tegata	/app/accounting/transactions/tegpay.nl
EDIT_TRAN_TEGPYBL	Issue Tegata	/app/accounting/transactions/tegpybl.nl
EDIT_TRAN_TEGRCVBL	Receive Tegata	/app/accounting/transactions/tegrcvbl.nl
EDIT_TRAN_TRANSFER	Transfer Funds	/app/accounting/transactions/transfer.nl
EDIT_TRAN_TRNFRORD	Enter Transfer Orders	/app/accounting/transactions/trnfrord.nl
EDIT_TRAN_UNBUILD	Unbuild Assemblies	/app/accounting/transactions/unbuild.nl
EDIT_TRAN_VATLIABAU	Pay Tax Liability	/app/accounting/transactions/vatliab.nl
EDIT_TRAN_VENDAUTH	Enter Vendor Return Authorizations	/app/accounting/transactions/vendauth.nl
EDIT_TRAN_VENDBILL	Enter Bills	/app/accounting/transactions/vendbill.nl
EDIT_TRAN_VENDCRED	Enter Vendor Credits	/app/accounting/transactions/vendcred.nl
EDIT_TRAN_VENDPYMT	Pay Single Vendor	/app/accounting/transactions/vendpymt.nl
EDIT_TRAN_VENDRFQ	Enter Vendor Requests For Quote	/app/accounting/transactions/vendrfq.nl
EDIT_TRAN_WITHDRAWINVENTORY	Withdraw By Transfer Order	/app/accounting/transactions/inventory/bulktransferorder/withdrawinventory.nl
EDIT_TRAN_WOCLOSE	Close Work Order	/app/accounting/transactions/woclose.nl
EDIT_TRAN_WOCOMPL	Complete Work Order	/app/accounting/transactions/wocompl.nl
EDIT_TRAN_WOISSUE	Issue Work Order	/app/accounting/transactions/woissue.nl
EDIT_TRAN_WORKORD	Enter Work Orders	/app/accounting/transactions/workord.nl
EDIT_TRAN_YTDADJST	Create Payroll Adjustment	/app/accounting/transactions/ytdadjst.nl
EDIT_UNITSTYPE	New Units Of Measure	/app/common/units/unitstype.nl
EDIT_UPGRADEBUNDLE	Managed Bundles	/app/bundler/bundlelist.nl?type=S&subtype=m
EDIT_URLALIAS	New Promotional URL	/app/setup/urlalias.nl
EDIT_USAGE	Create Usages	/app/accounting/otherlists/usage.nl
EDIT_VENDOR	New Vendors	/app/common/entity/vendor.nl



Task ID	Page Label in NetSuite	URL
EDIT_WEBAPPS	New SSP Application	/app/common/scripting/webapp.nl
EDIT_WEBAPPSADV	New SSP Application	/app/common/scripting/webapp.nl
EDIT_WORKCALENDAR	New Work Calendar	/app/accounting/project/workcalendar.nl
EDIT_WORKFLOW	New Workflow	/app/common/workflow/setup/workflowmanager.nl
EDIT_WORKPLACE	New Workplaces	/app/common/otherlists/workplacetype.nl
LIST_ACCOUNT	Accounts	/app/accounting/account/accounts.nl
LIST_ACCOUNTINGBOOK	Accounting Books	/app/accounting/multibook/accountingbooklist.nl
LIST_ACCOUNTINGOTHERLIST	Accounting Lists	/app/common/otherlists/accountingotherlists.nl
LIST_ACTIVITY	Activity List	/app/crm/calendar/activitylist.nl
LIST_ADVPDFTEMPLATE	Advanced PDF/HTML Templates	/app/common/custom/pdftemplates.nl?sc=-90
LIST_ALLOCATION	Allocation Schedules	/app/accounting/transactions/allocschedulelist.nl
LIST_ALLOCATIONBATCH	Allocation Batches	/app/accounting/transactions/allocationbatchlist.nl
LIST_AMORTIZATION	Amortization Templates	/app/accounting/otherlists/amortizationtemplates.nl
LIST_AMORTIZATIONSCHEDULE	Amortization Schedules	/app/accounting/otherlists/amortizationschedules.nl
LIST_APPDEF	App Definitions	/app/appdef/appdeflist.nl
LIST_APPPKG	App Packages	/app/appdef/apppkglist.nl
LIST_APPPUBLISHERS	Application Publishers	/app/setup/apppublishers.nl
LIST_APPROVEACH	Approve Direct Deposits	/app/accounting/transactions/approveach.nl
LIST_APPROVEEFT	Approve Electronic Funds Transfers	/app/accounting/transactions/approveeft.nl
LIST_APPROVERSRCALLOCATION	Approve Resource Allocations	/app/common/bulk/approval/recordapproval.nl?type=resourceallocation
LIST_APPROVERTIMEENTRY	Approve Time	/app/common/bulk/approval/recordapproval.nl?type=timeentry
LIST_APPROVEVP	Approve Vendor Payment Transfers	/app/accounting/transactions/approvevp.nl
LIST_BILLINGACCOUNT	Billing Accounts	/app/accounting/otherlists/billingaccounts.nl
LIST_BILLINGCLASS	Billing Classes	/app/accounting/otherlists/billingclasses.nl

Task ID	Page Label in NetSuite	URL
LIST_BILLINGRULE	Billing Rule	/app/accounting/transactions/billing/billingrules.nl
LIST_BILLINGRUNRESULTS	Billing Run Results	/app/accounting/transactions/billing/billingrunresults.nl
LIST_BILLINGSCHEDULE	Billing Schedules	/app/accounting/otherlists/billingschedules.nl
LIST_BILLING_GROUPS	Billing Groups	/app/crm/common/crmgroupelist.nl?billinggroupsonly=T
LIST_BILLING_WORK_CENTER	Billing Work Center	/app/accounting/transactions/billingworkcenter/billingworkcenter.nl
LIST_BILLOFDISTRIBUTION	Bill Of Distribution	/app/accounting/inventory/distributionplanning/billofdistributionlist.nl
LIST_BILLRUN	Billing Operations	/app/accounting/transactions/billingworkcenter/billrunresults.nl
LIST_BILLRUNSCHEDULE	Billing Operation Schedules	/app/accounting/transactions/billingworkcenter/billrunschedules.nl
LIST_BIN	Bins	/app/accounting/transactions/inventory/binlist.nl
LIST_BUDGET	Budgets	/app/accounting/transactions/budgetlist.nl
LIST_BUDGETRATES	Budget Exchange Rates	/app/accounting/otherlists/budgetrates.nl
LIST_BULKOP	Mass Updates	/app/common/bulk/bulkops.nl
LIST_BULKRESULTS	Mass Update Results	/app/common/bulk/bulkresults.nl
LIST_BUNDLE	List Bundles	/app/bundler/bundlelist.nl?type=S
LIST_CALENDAR	Calendar	/app/crm/calendar/calendar.nl
LIST_CALL	Phone Calls	/app/crm/calendar/calllist.nl
LIST_CAMPAIGN	Marketing Campaigns	/app/crm/marketing/campaignlist.nl?Campaign_ISSALESCAMPAIGN=F
LIST_CAMPAIGNAUDIENCE	Campaign Audiences	/app/crm/marketing/campaignaudiences.nl
LIST_CAMPAIGNCATEGORY	Campaign Categories	/app/crm/marketing/campaigncategories.nl
LIST_CAMPAIGNCHANNEL	Campaign Channels	/app/crm/marketing/campaignchannels.nl
LIST_CAMPAIGNEMAIL	Campaign Email Addresses	/app/crm/marketing/campaignemails.nl
LIST_CAMPAIGNFAMILY	Campaign Families	/app/crm/marketing/campaignfamilies.nl
LIST_CAMPAIGNOFFER	Campaign Offers	/app/crm/marketing/campaignoffers.nl
LIST_CAMPAIGNSEARCHENGINE	Campaign Search Engines	/app/crm/marketing/campaignsearchengines.nl

Task ID	Page Label in NetSuite	URL
LIST_CAMPAIGNSUBSCRIPTION	Campaign Subscriptions	/app/crm/marketing/campaignsubscriptions.nl
LIST_CAMPAIGNVERTICAL	Campaign Verticals	/app/crm/marketing/campaignverticals.nl
LIST_CASEFIELDRULE	Set Up Case Rules	/app/crm/support/casefieldrules.nl
LIST_CASEFORM	Online Case Forms	/app/crm/support/caseforms.nl
LIST_CASEISSUE	Case Issues	/app/crm/support/caseissuelist.nl
LIST_CASEORIGIN	Case Origin Types	/app/crm/support/caseoriginlist.nl
LIST_CASEPRIORITY	Case Priorities	/app/crm/support/caseprioritylist.nl
LIST_CASEPROFILE	Case Profile	/app/crm/support/profiles/caseprofiles.nl
LIST_CASESTATUS	Case Statuses	/app/crm/support/casestatuslist.nl
LIST_CASETERRITORIES	Case Territory List	/app/crm/support/supportterritorylist.nl
LIST_CASETERRITORY	Manage Case Territories	/app/crm/common/automation/territorymanager.nl?case=T
LIST_CASETERRITORYASSIGN	Territory Reassignment	/app/common/bulk/bulkop.nl?searchtype=Case&opcode=ReAssign
LIST_CASETYPE	Case Types	/app/crm/support/casetypelist.nl
LIST_CCTRAN	View Credit Card Transactions	/app/accounting/transactions/cardtrans.nl
LIST_CHARGE	Charges	/app/accounting/transactions/billing/charges.nl
LIST_CHARGERULE	View Charge Rules	/app/accounting/transactions/billing/chargerules.nl
LIST_CHARGERUNRESULTS	Charge Run Results	/app/accounting/transactions/billing/chargerunresults.nl
LIST_CHART_ACCOUNT	Chart of Accounts	/app/accounting/account/accounts.nl?report=T&code=COA
LIST_CLASS	Classes	/app/common/otherlists/classlist.nl
LIST_CLASSSEGMENTMAPPING	Class Mapping	/app/accounting/account/classsegmentmappinglist.nl
LIST_COLORTHEME	Color Themes	/app/setup/look/colorthemes.nl
LIST_COMMERCECATALOG	Commerce Catalogs	/app/platform/services/commercecategory/commercecatalogs.nl
LIST_COMMERCECATEGORY	Commerce Categories	/app/platform/services/commercecategory/commercecategories.nl
LIST_COMMISSIONSCHEDULE	Employee Schedules	/app/crm/sales/commissions/commissionscheds.nl
LIST_COMPANY	All Companies	/app/common/entity/companylist.nl

Task ID	Page Label in NetSuite	URL
LIST_COMPETITOR	Competitors	/app/crm/sales/competitorlist.nl
LIST_CONSOLRATES	Consolidated Exchange Rates	/app/accounting/otherlists/consolidatedrates.nl
LIST_CONTACT	Contacts	/app/common/entity/contactlist.nl
LIST_COUNTRY	Set Up Countries	/app/setup/countries.nl
LIST_CRMGROUP	Groups	/app/crm/common/crmgroupelist.nl
LIST_CRMOTHERLIST	CRM Lists	/app/common/otherlists/crmotherlists.nl
LIST_CRMTEMPLATE	Marketing Templates	/app/crm/common/merge/marketingtemplates.nl
LIST_CSPSETUP	Setup CSP	/app/setup/CSPList.nl
LIST_CURRENCY	Currencies	/app/common/multicurrency/currencylist.nl
LIST_CURRENCYRATE	Currency Exchange Rates	/app/common/multicurrency/currencyratelist.nl
LIST_CUSTADDRESSENTRYFORM	Address Forms	/app/common/custom/custaddressentryforms.nl
LIST_CUSTBODYFIELD	Transaction Body Fields	/app/common/custom/bodycustfields.nl
LIST_CUSTCATEGORY	Center Categories	/app/common/custom/custcategories.nl
LIST_CUSTCENTER	Centers	/app/common/custom/custcenters.nl
LIST_CUSTCOLUMNFIELD	Transaction Column Fields	/app/common/custom/columncustfields.nl
LIST_CUSTMAILAYOUT	Transaction Form HTML Layouts	/app/common/custom/custemaillayouts.nl
LIST_CUSTENTITYFIELD	Entity Fields	/app/common/custom/entitycustfields.nl
LIST_CUSTENTRYFORM	Entry Forms	/app/common/custom/custentryforms.nl
LIST_CUSTEVENTFIELD	CRM Fields	/app/common/custom/eventcustfields.nl
LIST_CUSTFIELDTAB	Subtabs	/app/common/custom/custfieldtabs.nl
LIST_CUSTFORM	Transaction Forms	/app/common/custom/custforms.nl
LIST_CUSTITEMFIELD	Item Fields	/app/common/custom/itemcustfields.nl
LIST_CUSTITEMNUMBERFIELD	Item Number Fields	/app/common/custom/itemnumbercustfields.nl
LIST_CUSTJOB	Customers	/app/common/entity/custjoblist.nl?Customer_STAGE=CUSTOMER
LIST_CUSTLAYOUT	Transaction Form PDF Layouts	/app/common/custom/custlayouts.nl
LIST_CUSTLIST	Lists	/app/common/custom/custlists.nl



Task ID	Page Label in NetSuite	URL
LIST_CUSTOMCODEFILES	SuiteScripts	/app/common/media/mediaitemfolders.nl?folder=-15
LIST_CUSTOMERFIELDRULE	Set Up Sales Rules	/app/crm/sales/customerfieldrules.nl
LIST_CUSTOMERFORM	Online Customer Forms	/app/crm/sales/leadforms.nl
LIST_CUSTOMERSTATUS	Customer Statuses	/app/crm/sales/customerstatuslist.nl
LIST_CUSTOMSEGMENT	Custom Segments	/app/common/custom/segments/segments.nl
LIST_CUSTOMSUBLIST	Sublists	/app/common/custom/customsublists.nl
LIST_CUSTOTHERFIELD	Other Custom Fields	/app/common/custom/othercustfields.nl
LIST_CUSTRECORD	Record Types	/app/common/custom/custrecords.nl
LIST_CUSTRECORDFORM	Online Custom Record Forms	/app/common/custom/custrecordforms.nl
LIST_CUSTSECTION	Center Tabs	/app/common/custom/custsections.nl
LIST_CUSTTAB	Custom Subtabs	/app/common/custom/subtabs.nl
LIST_CUSTTRANSACTION	Transaction Types	/app/common/custom/customtransactions.nl
LIST_DEPARTMENT	Departments	/app/common/otherlists/departmentlist.nl
LIST_DEPTSEGMENTMAPPING	Department Mapping	/app/accounting/account/deptsegmentmappinglist.nl
LIST_DEVICE_ID	Device ID	/app/setup/deviceauth/devicelist.nl
LIST_DIFFROLE	Show Role Differences	/app/setup/diffroles.nl
LIST_DISTRIBUTIONNETWORK	Distribution Network	/app/accounting/inventory/distributionplanning/distributionnetworklist.nl
LIST_DRIVERSLICENSE	Driver's Licences	/app/hcm/hris/governmentid/driverslicenses.nl
LIST_DUPLICATE_RESOLUTION_STATUS	Duplicate Resolution Status	/app/common/entity/duplicatemanagement/dupejob.nl
LIST_EMAILTEMPLATE	Email Templates	/app/crm/common/merge/emailtemplates.nl
LIST_EMPLCATEGORY	Employee Directory	/app/site/setup/emplcategories.nl
LIST_EMPLOYEE	Employees	/app/common/entity/employeelist.nl
LIST_EMPOOTHERLIST	Employee Related Lists	/app/common/otherlists/empotherlists.nl
LIST_ENTITY	All Entities	/app/common/entity/entitylist.nl
LIST_ENTITYACCOUNTMAPPING	Entity Account Mappings	/app/accounting/account/entityaccountmappinglist.nl
LIST_ENTITY_DUPLICATES	Entity Duplicate Resolution	/app/common/entity/manageduplicates.nl



Task ID	Page Label in NetSuite	URL
LIST_ESCALATIONRULE	Set Up Escalation Rules	/app/crm/support/escalationfieldrules.nl
LIST_ESCALATIONTERRITORY	Manage Escalation Assignment	/app/crm/common/automation/territorymanager.nl?escalation=T
LIST_EVENT	Events	/app/crm/calendar/eventlist.nl
LIST_EXPCATEGORY	Expense Categories	/app/accounting/otherlists/expcategories.nl
LIST_FAIRVALUEDIMENSION	Fair Value Dimensions	/app/accounting/revrec/fairvaluedimension.nl
LIST_FAIRVALUEFORMULA	Fair Value Formulas	/app/accounting/revrec/fairvalueformulalist.nl
LIST_FAIRVALUEPRICE	Fair Value Price Lists	/app/accounting/revrec/fairvaluepricelist.nl
LIST_FAXTEMPLATE	Fax Templates	/app/crm/common/merge/faxtemplates.nl
LIST_FCSITEFOLDER	Web Site Hosting Files	/app/common/media/mediaitemfolders.nl?folder=-100
LIST_FCSITEFOLDERADV	Web Site Hosting Files	/app/common/media/mediaitemfolders.nl?folder=-100
LIST_FILECABINET	Media Items	/app/common/media/mediaitems.nl
LIST_FINCHRG	Assess Finance Charges	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustInv&Transaction_FINCHRG=T
LIST_FISCALCALENDAR	Fiscal Calendars	/app/setup/period/fiscalcalendars.nl
LIST_FORECAST	List Sales Rep Forecast	/app/crm/sales/forecastlist.nl?Forecast_ISTEAM=F
LIST_GENERICRESOURCE	Generic Resources	/app/accounting/project/genericresources.nl
LIST_GIFTCERTIFICATES	Gift Certificates	/app/accounting/otherlists/giftcertificates.nl
LIST_GLNUMHISTORY	GL Audit Numbering History	/app/accounting/transactions/glnumbering/glnumhistory.nl
LIST_GLOBALACCOUNTAPPING	Global Account Mappings	/app/accounting/account/globalaccountmappinglist.nl
LIST_GOVERNMENTISSUEDIDTYPE	Government-Issued ID Types	/app/hcm/hris/governmentid/governmentissuedidtypes.nl
LIST_HCMJOB	Jobs	/app/hcm/hris/jobs.nl
LIST_IMAGE	Images	/app/common/media/mediaitemfolders.nl?folder=-4
LIST_INFOITEM	Information Items	/app/site/setup/infoitemlist.nl?searchid=-2540
LIST_INFOITEMFORM	Publish Forms	/app/site/setup/infoitemlist.nl?searchid=-2541



Task ID	Page Label in NetSuite	URL
LIST_INSTALLEDBUNDLE	Installed Bundles	/app/bundler/bundlelist.nl?type=l
LIST_INVCOSTTEMPLATE	Inventory Cost Template	/app/accounting/inventory/standard/invcosttemplatelist.nl
LIST_ISSUE	Issues	/app/crm/support/issuedb/issuelist.nl
LIST_ITEM	Items	/app/common/item/itemlist.nl
LIST_ITEMACCOUNTMAPPING	Item Account Mappings	/app/accounting/account/itemaccountmappinglist.nl
LIST_ITEMATTRGROUP	Field Sets	/app/site/setup/fieldsets.nl
LIST_ITEMDEMANDPLAN	Item Demand Plans	/app/accounting/inventory/demandplanning/itemdemandplanlist.nl
LIST_ITEMOPTION	Transaction Item Options	/app/common/custom/itemoptions.nl
LIST_ITEMREVENUECATEGORY	Item Revenue Categories	/app/accounting/revrec/itemrevenuecategorylist.nl
LIST_ITEMSUPPLYPLAN	Item Supply Plans	/app/accounting/inventory/demandplanning/itemsupplyplanlist.nl
LIST_ITEM_REVISION	Item Revisions	/app/common/item/itemrevisionlist.nl
LIST_JOB	Jobs	/app/accounting/project/projects.nl
LIST_JOBREQUISITION	Job Requisitions	/app/hcm/hris/jobrequisitions.nl
LIST_KBCATEGORY	Knowledge Base	/app/site/setup/kbcategories.nl
LIST_KPIREPORT	KPI Scorecards	/app/center/enhanced/kpireports.nl
LIST_KUDOS	Kudos	/app/hcm/perfmgmt/kudoslist.nl
LIST_LEAD	Leads	/app/common/entity/custjoblist.nl?Customer_STAGE=LEAD
LIST_LOCASSIGNCONF	Configurations List	/app/setup/locationassignment/configuration/locationassignmentconfigurationlist.nl
LIST_LOCATION	Locations	/app/common/otherlists/locationlist.nl
LIST_LOCATIONCOSTINGGROUP	Location Costing Groups	/app/accounting/inventory/costing/locationcostinggrouplist.nl
LIST_LOCSEGMENTMAPPING	Location Mapping	/app/accounting/account/locsegmentmappinglist.nl
LIST_MAILMERGE	Merge History	/app/crm/common/merge/mailmergehistory.nl
LIST_MAILTEMPLATE	Letter Templates	/app/crm/common/merge/mailtemplates.nl
LIST_MAPREDUCESCRIPTSTATUS	Map/Reduce Script Status	/app/common/scripting/mapreducescriptstatus.nl
LIST_MEDIAITEMFOLDER	File Cabinet	/app/common/media/mediaitemfolders.nl
LIST_MEDIAITEMFOLDER_LOG	Job Status	/app/external/xml/upload/uploadlog.nl?displayType=FILECABINET



Task ID	Page Label in NetSuite	URL
LIST_MEMDOC	Memorized Transactions	/app/accounting/otherlists/memdochist.nl
LIST_MEMDOCRESULTS	Memorized Transactions Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=MEMORIZEDTRANSACTIONS&BulkProcSubmission_CREATEDDATE=TODAY
LIST_MFGCOSTTEMPLATE	Manufacturing Cost Templates	/app/accounting/manufacturing/mfgcosttemplatelist.nl
LIST_MFGOPERATIONTASK	Manufacturing Operation Tasks	/app/accounting/manufacturing/mfgoperationlist.nl
LIST_MFGROUTING	Manufacturing Routing	/app/accounting/manufacturing/mfgroutinglist.nl
LIST_MGRFORECAST	List Sales Manager Forecast	/app/crm/sales/forecastlist.nl?Forecast_ISTEAM=T
LIST_MYROLES	My Roles	/app/center/myroles.nl
LIST_NOTIFICATION	Notifications	/app/email/alerts/notifications.nl
LIST_ORGANIZATIONVALUE	Organization Values	/app/hcm/perfmgmt/organizationvalue.list.nl
LIST_OTHERGOVERNMENTISSUEDID	Other Government-Issued IDs	/app/hcm/hris/governmentid/othergovernmentissuedids.nl
LIST_OTHERNAME	Other Names	/app/common/entity/othernames.nl
LIST_PARTNER	Partners	/app/common/entity/partnerlist.nl
LIST_PARTNERCOMMISSIONSCHED	Partner Schedules	/app/crm/sales/commissions/partnercommissionscheds.nl
LIST_PARTNERPLANASSIGN	Partner Plans	/app/crm/sales/commissions/partnercommissionplans.nl
LIST_PARTNERPLANASSIGNS	Partner Plan Assignments	/app/crm/sales/commissions/partnerplanassigns.nl
LIST_PASSPORT	Passports	/app/hcm/hris/governmentid/passports.nl
LIST_PAYMENTEVENT	View Payment Events	/app/paymentprocessing/paymentevents.nl
LIST_PAYMETHODS	Payment Methods	/app/common/otherlists/accountingotherlists.nl?tname=paymeth
LIST_PAYPALACCOUNT	PayPal Accounts	/app/external/paypal/paypalaccountlist.nl
LIST_PAYROLLBATCH	Payroll Batches	/app/payroll/payrollbatchlist2.nl
LIST_PAYROLLISSUES	Payroll Issues	/app/payroll/cdstatus.nl
LIST_PAYROLLITEM	Payroll Items	/app/common/item/payrollitems.nl
LIST_PAYTERMS	Payment Terms	/app/common/otherlists/accountingotherlists.nl?tname=terms

Task ID	Page Label in NetSuite	URL
LIST_PDFTEMPLATE	PDF Templates	/app/crm/common/merge/pdftemplates.nl
LIST_PERIOD	Manage Accounting Periods	/app/setup/period/fiscalperiods.nl
LIST_PLANASSIGN	Employee Plans	/app/crm/sales/commissions/commissionplans.nl
LIST_PLANASSIGNS	Employee Plan Assignments	/app/crm/sales/commissions/planassigns.nl
LIST_PLANNEDSTANDARD COST	Planned Standard Costs	/app/accounting/inventory/plannedstandardcostlist.nl
LIST_PLUGIN	Plug-in Implementations	/app/common/scripting/pluginlist.nl
LIST_PLUGINTYPE	Custom Plug-in Types	/app/common/scripting/plugintypelist.nl
LIST_POSITION	Positions	/app/hcm/hris/positions.nl
LIST_PRESCATEGORY	Categories	/app/site/setup/prescategories.nl?ctype=PRES&section=&siteid=
LIST_PROCESSHISTTXNSTATUS	Historical Transaction Processing Status	/app/accounting/multibook/htp/status.nl
LIST_PROJECTEXPENSETYPE	Project Expense Types	/app/accounting/otherlists/projectexpensetypes.nl
LIST_PROJECTTASK	Project Tasks	/app/accounting/project/projecttasks.nl
LIST_PROJECTTASKIMPORT	Import Project Tasks	/app/setup/assistants/nsimport/simpleimport.nl?rectype=PROJECTTASK
LIST_PROJECTTEMPLATE	Project Templates	/app/accounting/project/projecttemplates.nl
LIST_PROSPECT	Prospects	/app/common/entity/custjoblist.nl?Customer_STAGE=PROSPECT
LIST_PUBLISHERAPP	List Applications	/app/suiteapp/devframework/appregistrylist.nl
LIST_QUANTITYPRICINGSCHEDULE	Quantity Pricing Schedules	/app/accounting/otherlists/quantitypricingschedules.nl
LIST_QUOTA	Quotas	/app/crm/sales/quotalist.nl
LIST_RATEPLAN	Rate Plans	/app/accounting/otherlists/rateplans.nl
LIST_RECENTRECORDS	Recent Records	/app/common/otherlists/recentrecords.nl
LIST_RECVFILES	Attachments Received	/app/common/media/mediaitemfolders.nl?folder=-10
LIST_REFERRALCODE	Promotions	/app/crm/sales/referralcodelist.nl
LIST_REGION	Set Up Regions	/app/setup/region/regionlist.nl
LIST RELATEDITEM	Related Items Categories	/app/site/setup/relateditems.nl
LIST RELATEDITEMADV	Related Items Categories	/app/site/setup/relateditems.nl
LIST_REPORESULT	Report Results	/app/reporting/workqueue/reportresultlist.nl

Task ID	Page Label in NetSuite	URL
LIST_REPO_SCHEDULE	Report Schedules	/app/reporting/workqueue/reportschedulelist.nl
LIST_RESOURCE	Resources	/app/crm/calendar/resources.nl
LIST_REVENUEALLOCATINGROUP	Revenue Allocation Groups	/app/common/otherlists/accountingotherlists.nl?tname=revenueallocationgroup
LIST_REVENUEELEMENT	Revenue Elements	/app/accounting/revrec/revenueelementlist.nl
LIST_REVENUEPLAN	Revenue Recognition Plans	/app/accounting/revrec/revenueplanlist.nl
LIST_REVNUERECOGNITIONRULE	Revenue Recognition Rules	/app/accounting/revrec/revenuerecognitionrulelist.nl
LIST_REVRECFIELDMAPPING	Revenue Field Mapping	/app/accounting/revrec/revrecfieldmapping.nl
LIST_REVRECSCHEDULES	Revenue Recognition Schedules	/app/accounting/otherlists/revreccschedules.nl
LIST_REVRECSCHEDULE	Revenue Recognition Templates	/app/accounting/otherlists/revrectemplates.nl
LIST_ROLE	Manage Roles	/app/setup/rolelist.nl
LIST_RSRCALLOCATION	Rsrc Allocations	/app/accounting/project/allocations.nl
LIST_RSSFEED	RSS Feeds	/app/site/hosting/rssfeeds.nl
LIST_SALESCAMPAIGN	Sales Campaigns	/app/crm/marketing/salescampaignlist.nl
LIST_SALESTEAM	Sales Teams	/app/crm/common/salesteamlist.nl
LIST_SALESTERRITORIES	Sales Territory List	/app/crm/sales/salesterritorylist.nl
LIST_SALESTERRITORY	Manage Sales Territories	/app/crm/common/automation/territorymanager.nl?sales=T
LIST_SALESTERRITORYASSIGN	Territory Reassignment	/app/common/bulk/bulkop.nl?searchtype=Customer&opcode=ReAssign
LIST_SAVEDASHBOARD	Published Dashboards	/app/center/setup/savedashboards.nl
LIST_SAVEDBULKOP	Saved Mass Updates	/app/common/bulk/savedbulkops.nl
LIST_SAVEDSEARCH	Saved Searches	/app/common/search/savedsearchlist.nl
LIST_SCRIPT	Scripts	/app/common/scripting/scriptlist.nl
LIST_SCRIPTEDRECORD	Scripted Records	/app/common/scripting/scriptedrecords.nl
LIST_SCRIPTLOGS	Script Execution Logs	/app/common/scripting/scriptnotearchive.nl
LIST_SCRIPTRECORD	Script Deployments	/app/common/scripting/scriptrecordlist.nl
LIST_SCRIPTSTATUS	Scheduled Script Status	/app/common/scripting/scriptstatus.nl

Task ID	Page Label in NetSuite	URL
LIST_SEARCHRESULTS	Search Results	/app/common/search/searchresults.nl
LIST_SEARCHRESULTSARCHIVE	Search Results	/app/common/search/backend/searchresultsarchive.nl
LIST_SENTFILES	Attachments Sent	/app/common/media/mediaitemfolders.nl?folder=-14
LIST_SHIPITEM	Shipping Items	/app/common/item/shipitems.nl
LIST_SHIPPARTREGISTRATION	Shipping Partner Registrations	/app/setup/shipping.nl?tab=partnerregistrations
LIST_SHIPPINGMANIFEST	Shipping Manifest	/app/common/shipping/fedex/shippingmanifest.nl
LIST_SITEMAILTEMPLATE	Web Store Email Templates	/app/site/setup/siteemailtemplates.nl
LIST_SITEITEMTEMPLATE	Item/Category Templates	/app/site/setup/siteitemtemplates.nl
LIST_SITETAGS	Web Site Tags	/app/site/setup/sitetags.nl
LIST_SITETHemes	Web Site Themes	/app/site/setup/sitethemes.nl
LIST_SMBIMPORT	SMB Import	/app/external/xml/upload/upload.nl
LIST_SMTP_SERVERS	SMTP Servers	/app/setup/smtpservers.nl
LIST SOLUTION	Solutions	/app/crm/support/kb/solutions.nl
LIST_SS_CATEGORY	Publish Saved Search	/app/site/setup/sscategories.nl
LIST_STANDARDCOSTVERSION	Standard Cost Versions	/app/accounting/inventory/standardcostversionlist.nl
LIST_STATE	Set Up States/Provinces/Counties	/app/setup/states.nl
LIST_STATUSACH	View Direct Deposit Status	/app/accounting/transactions/statusach.nl
LIST_STATUSEFT	View Electronic Funds Transfer Status	/app/accounting/transactions/statuseft.nl
LIST_STATUSVP	View Vendor Payment Status	/app/accounting/transactions/statusvp.nl
LIST_STOREITEMLISTLAYOUT	Layouts	/app/site/setup/storeitemlistlayouts.nl
LIST_STORETAB	Tabs	/app/site/setup/storetabs.nl
LIST_SUBSCRIPTION	Subscriptions	/app/accounting/subscription/subscriptionlist.nl
LIST_SUBSCRIPTIONCHANGEORDER	View Subscription Change Orders	/app/accounting/subscription/subscriptionchangeorders.nl
LIST_SUBSCRIPTIONCHANGEORDER2	Subscription Change Orders	/app/accounting/subscription/subscriptionchangeorders.nl
LIST_SUBSCRIPTIONONLINE	Subscription Lines	/app/accounting/subscription/subscriptionlinelist.nl
LIST_SUBSCRIPTIONPLAN	Subscription Plans	/app/accounting/subscription/subscriptionplans.nl



Task ID	Page Label in NetSuite	URL
LIST_SUBSCRIPTIONRENEWALHISTORY	View Subscription Renewal History	/app/accounting/subscription/autorenewal/subscriptionrenewalhistory.nl
LIST_SUBSCRIPTIONRENEWALHISTORY2	Subscription Renewal History	/app/accounting/subscription/autorenewal/subscriptionrenewalhistory.nl
LIST_SUBSIDIARY	Subsidiaries	/app/common/otherlists/subsidiarylist.nl
LIST_SUPPORTCASE	Cases	/app/crm/support/caselist.nl
LIST_SYSALERT	System Alerts	/app/common/otherlists/systemalerts.nl
LIST_SYSTEMEMAILTEMPLATE	Set Up System Email Templates	/app/crm/common/merge/systememailtemplates.nl
LIST_TASK	Tasks	/app/crm/calendar/tasklist.nl
LIST_TAXGROUP	Tax Groups	/app/common/item/taxgroups.nl
LIST_TAXITEM	Tax Codes	/app/common/item/taxitems.nl
LIST_TAXPERIOD	Manage Tax Periods	/app/setup/period/taxperiods.nl
LIST_TAXSCHEDULE	Tax Schedules	/app/common/item/taxschedules.nl
LIST_TEMPLATEFILES	Template Files	/app/common/media/mediaitemfolders.nl?folder=-9
LIST_TEMPLATE_CATEGORY	Template Categories	/app/crm/common/merge/templatecategories.nl
LIST_TERMINATIONREASON	Termination Reasons	/app/hcm/hris/terminationreasons.nl
LIST_TIMEBILL_WEEKLY	List Weekly Time Tracking	/app/accounting/transactions/weeklytimelist.nl
LIST_TIMEOFFCHANGE	Time-Off Changes	/app/hcm/hris/timeoffmanagement/timeoffchanges.nl
LIST_TIMEOFFPLAN	Time-Off Plans	/app/hcm/hris/timeoffmanagement/timeoffplans.nl
LIST_TIMEOFFREQUEST	Time-Off Requests	/app/hcm/hris/timeoffmanagement/timeoffrequests.nl
LIST_TIMEOFFTYPE	Time-Off Types	/app/hcm/hris/timeoffmanagement/timeofftypes.nl
LIST_TIMESHEET	Timesheets	/app/accounting/transactions/timesheets.nl
LIST_TOPIC	Topics	/app/crm/support/kb/topics.nl
LIST_TRANNUMBERAUDITLOG	Transaction Numbering Audit Log	/app/accounting/transactions/tranumberauditlist.nl
LIST_TRANSACTION	Transactions	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=@ALL@

Task ID	Page Label in NetSuite	URL
LIST_TRAN_BINTRNFR	Bin Transfers	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=BinTrnfr
LIST_TRAN_BINWKSHT	Bin Putaway Worksheets	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=BinWksht
LIST_TRAN_BLANKORD	Blanket Purchase Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=BlankOrd
LIST_TRAN_BOOKICJOURNAL	Accounting Book Intercompany Journal Entries	/app/accounting/transactions/transactionlistswitch.nl?TR_Transaction_TYPE=Journal&icj=T
LIST_TRAN_BOOKJOURNAL	Accounting Book Journal Entries	/app/accounting/transactions/transactionlistswitch.nl?TR_Transaction_TYPE=Journal
LIST_TRAN_BUILD	Assembly Builds	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Build
LIST_TRAN_CARDCHRG	Credit Card Charges	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CardChrg
LIST_TRAN_CASHRFND	Cash Refunds	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CashRfnd
LIST_TRAN_CASHSALE	Cash Sales	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CashSale
LIST_TRAN_CHECK	Checks	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Check
LIST_TRAN_COMMISSN	Individual Employee Commissions	/app/accounting/transactions/commissns.nl
LIST_TRAN_CUSTCHRG	Statement Charges	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustChrg
LIST_TRAN_CUSTCRED	Credit Memos	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustCred
LIST_TRAN_CUSTDEP	Customer Deposits	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustDep
LIST_TRAN_CUSTINVC	Invoices	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustInvc

Task ID	Page Label in NetSuite	URL
LIST_TRAN_CUSTPYMT	Customer Payments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustPymt
LIST_TRAN_CUSTRFND	Customer Refunds	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=CustRfnd
LIST_TRAN_DEPAPPL	Deposit Applications	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=DepAppl
LIST_TRAN_DEPOSIT	Deposits	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Deposit
LIST_TRAN_DOWNLOAD	My Downloads	/app/common/entity/downloads.nl
LIST_TRAN_ESTIMATE	Estimates	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Estimate
LIST_TRAN_EXPREPT	Expense Reports	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ExpRept
LIST_TRAN_FFTREQ	Manage Fulfillment Requests	/app/accounting/transactions/fulfillmentrequest/fftreqlist.nl?Transaction_TYPE=FftReq
LIST_TRAN_FXREVAL	Currency Revaluations	/app/accounting/transactions/transactionlistswitch.nl?TR_Transaction_TYPE=FxReval
LIST_TRAN_ICJOURNAL	Intercompany Journal Entries	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Journal&icj=T
LIST_TRANICTRNFRORD	Intercompany Transfer Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TrnfrOrd&icto=T
LIST_TRAN_INVADJST	Inventory Adjustments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvAdjst
LIST_TRAN_INVCOUNT	Inventory Counts	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvCount
LIST_TRAN_INVDISTR	Inventory Distributions	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvDistr
LIST_TRAN_INVREVAL	Inventory Cost Revaluations	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvReval
LIST_TRAN_INVTRNFR	Inventory Transfers	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvTrnfr

Task ID	Page Label in NetSuite	URL
LIST_TRAN_INVWKSHT	Inventory Worksheets	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=InvWksht
LIST_TRAN_ITEMRCPT	Item Receipts	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ItemRcpt
LIST_TRAN_ITEMSHIP	Item Fulfillments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=ItemShip
LIST_TRAN_JOURNAL	Journal Entries	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Journal
LIST_TRAN_LIABPYMT	Liability Payments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=LiabPymt
LIST_TRAN_OPPRTNTY	Opportunities	/app/accounting/transactions/opprtntylist.nl
LIST_TRAN_PARTNERCOMMISSN	Individual Partner Commissions	/app/accounting/transactions/partnercommissns.nl
LIST_TRAN_PAYCHECK	Paychecks	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Paycheck
LIST_TRAN_PCHKJRNL	Paycheck Journals	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PChkJrn
LIST_TRAN_PURCHCON	Purchase Contracts	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchCon
LIST_TRAN_PURCHORD	Purchase Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchOrd
LIST_TRAN_PURCHORD_REQ	View Purchase Requests/Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchOrd
LIST_TRAN_PURCHREQ	Requisitions	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=PurchReq
LIST_TRAN_REORDER	Items Ordered	/app/common/item/itemsordered.nl
LIST_TRAN_REVARRNG	Revenue Arrangements	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=RevArrng
LIST_TRAN_REVCOMM	Revenue Commitments	/app/accounting/transactions/transactionlistswitch.nl?TR_Transaction_TYPE=RevComm

Task ID	Page Label in NetSuite	URL
LIST_TRAN_REVCOMRV	View Revenue Commitment Reversals	/app/accounting/transactions/transactionlistswitch.nl?TR_Transaction_TYPE=RevComRv
LIST_TRAN_REVCONTR	View Revenue Contracts	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=RevContr
LIST_TRAN_RFQ	Requests For Quote	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Rfq
LIST_TRAN_RTNAUTH	Return Authorizations	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=RtnAuth
LIST_TRAN_SALESORD	Sales Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=SalesOrd
LIST_TRAN_STATJOURNAL	Statistical Journal Entries	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Journal&statistical=T
LIST_TRAN_STATSCHEDULE	Statistical Schedules	/app/accounting/transactions/statistical/statisticschedulelist.nl
LIST_TRAN_STPICKUP	Store Pickup Fulfillments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=StPickUp
LIST_TRAN_TAXLIAB	Tax Liabilities	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab
LIST_TRAN_TAXPYMT	Tax Payments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxPyMt
LIST_TRAN_TEGPYBL	Issued Tegatas	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TegPybl
LIST_TRAN_TEGRCVBL	Received Tegatas	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TegRcvbl
LIST_TRAN_TRANSFER	Bank Transfers	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Transfer
LIST_TRAN_TRNFRORD	Transfer Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TrnfrOrd
LIST_TRAN_UNBUILD	Assembly Unbuilds	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=Unbuild
LIST_TRAN_VATLIAB	Tax Liabilities	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab

Task ID	Page Label in NetSuite	URL
LIST_TRAN_VATLIABAU	GST Liabilities	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab
LIST_TRAN_VATLIABUK	VAT Liabilities	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=TaxLiab
LIST_TRAN_VENDAUTH	Vendor Return Authorizations	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendAuth
LIST_TRAN_VENDBILL	Bills	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendBill
LIST_TRAN_VENDCRED	Bill Credits	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendCred
LIST_TRAN_VENDPYMT	Bill Payments	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendPymt
LIST_TRAN_VENDRFQ	Vendor Requests For Quote	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=VendRfq
LIST_TRAN_WOCLOSE	Work Order Closes	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=WOClose
LIST_TRAN_WOCOMPL	Work Order Completions	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=WOCompl
LIST_TRAN_WOISSUE	Work Order Issues	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=WOIssue
LIST_TRAN_WORKORD	Work Orders	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=WorkOrd
LIST_TRAN_YTDADJST	Payroll Adjustment	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=YtdAdjst
LIST_UNCATSITEITEM	View Uncategorized Items	/app/setup/uncatsiteitems.nl
LIST_UNCATSITEITEMADV	View Uncategorized Items	/app/setup/uncatsiteitems.nl
LIST_UNITSTYPE	Units Of Measure	/app/common/units/unitstypelist.nl
LIST_UPSELLPOPUP	Upsell Popup	/app/crm/sales/upsell/upsellpopup.nl
LIST_UPSELLWIZARD	Upsell Manager	/app/crm/sales/upsell/upsellmanager.nl
LIST_URLALIASES	Promotional URLs	/app/setup/urlaliases.nl
LIST_USAGE	Usages	/app/accounting/otherlists/usages.nl
LIST_USER	Manage Users	/app/setup/listusers.nl

Task ID	Page Label in NetSuite	URL
LIST_VENDOR	Vendors	/app/common/entity/vendorlist.nl
LIST_WEBAPPS	SSP Applications	/app/common/scripting/webapplist.nl
LIST_WEBAPPSADV	SSP Applications	/app/common/scripting/webapplist.nl
LIST_WORKCALENDAR	Work Calendars	/app/accounting/project/workcalendars.nl
LIST_WORKFLOW	Workflows	/app/common/workflow/setup/workflowlist.nl
LIST_WORKPLACE	Workplaces	/app/common/otherlists/workplaceclist.nl
REPO_10	ALT_SALES Pipeline by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=10
REPO_100	Total Pipeline by Sales Rep	/app/reporting/reportrunner.nl?cr=100
REPO_101	Total Pipeline by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=101
REPO_102	New Customer Sales Detail	/app/reporting/reportrunner.nl?cr=102
REPO_103	New Customer Sales	/app/reporting/reportrunner.nl?cr=103
REPO_104	Opportunities Lost	/app/reporting/reportrunner.nl?cr=104
REPO_105	Opportunities Won	/app/reporting/reportrunner.nl?cr=105
REPO_106	Sales Activity by Customer	/app/reporting/reportrunner.nl?cr=106
REPO_107	Sales Activity by Customer Detail	/app/reporting/reportrunner.nl?cr=107
REPO_108	Comparative Sales	/app/reporting/reportrunner.nl?cr=108
REPO_109	Income Statement	/app/reporting/reportrunner.nl?cr=109
REPO_11	ALT_SALES Pipeline by Sales Rep Summary	/app/reporting/reportrunner.nl?cr=11
REPO_110	Income Statement	/app/reporting/reportrunner.nl?cr=110
REPO_111	Authorized Commission	/app/reporting/reportrunner.nl?cr=111
REPO_112	Authorized Commission Detail	/app/reporting/reportrunner.nl?cr=112
REPO_113	GST on Purchases	/app/reporting/reportrunner.nl?cr=113
REPO_114	GST on Sales	/app/reporting/reportrunner.nl?cr=114
REPO_115	GST on Purchases Detail	/app/reporting/reportrunner.nl?cr=115
REPO_116	GST on Sales Detail	/app/reporting/reportrunner.nl?cr=116
REPO_118	Commissions Pending Authorization	/app/reporting/reportrunner.nl?cr=118
REPO_119	Commissions Pending Authorization Detail	/app/reporting/reportrunner.nl?cr=119
REPO_12	ALT_SALES Forecast by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=12
REPO_121	Estimated Commission	/app/reporting/reportrunner.nl?cr=121

Task ID	Page Label in NetSuite	URL
REPO_122	Estimated Commission Detail	/app/reporting/reportrunner.nl?cr=122
REPO_123	Campaign Response Detail	/app/reporting/reportrunner.nl?cr=123
REPO_126	Pipeline by Customer	/app/reporting/reportrunner.nl?cr=126
REPO_127	Pipeline by Customer Detail	/app/reporting/reportrunner.nl?cr=127
REPO_128	Pipeline by Sales Rep	/app/reporting/reportrunner.nl?cr=128
REPO_129	Pipeline ROI by Manager	/app/reporting/reportrunner.nl?cr=129
REPO_13	ALT_SALES Forecast by Sales Rep	/app/reporting/reportrunner.nl?cr=13
REPO_130	Pipeline by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=130
REPO_131	Pipeline by Status Summary	/app/reporting/reportrunner.nl?cr=131
REPO_132	Campaign ROI Analysis Detail	/app/reporting/reportrunner.nl?cr=132
REPO_133	Campaign Response	/app/reporting/reportrunner.nl?cr=133
REPO_134	Sales Activity by Sales Rep	/app/reporting/reportrunner.nl?cr=134
REPO_135	Sales Activity by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=135
REPO_14	Billings Forecast vs. Quota	/app/reporting/reportrunner.nl?cr=14
REPO_141	Partner Activity	/app/reporting/reportrunner.nl?cr=141
REPO_142	Partner Activity Detail	/app/reporting/reportrunner.nl?cr=142
REPO_143	Sales by Partner	/app/reporting/reportrunner.nl?cr=143
REPO_144	Sales by Partner Detail	/app/reporting/reportrunner.nl?cr=144
REPO_145	Shipping Report	/app/reporting/reportrunner.nl?cr=145
REPO_146	Total Open Opportunities Detail	/app/reporting/reportrunner.nl?cr=146
REPO_147	Forecast vs. Quota	/app/reporting/reportrunner.nl?cr=147
REPO_148	Sales Orders Pending Fulfillment	/app/reporting/reportrunner.nl?cr=148
REPO_149	Forecast by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=149
REPO_15	ALT_SALES and Billings Forecast vs. Quota	/app/reporting/reportrunner.nl?cr=15
REPO_150	Inventory Back Order Report	/app/reporting/reportrunner.nl?cr=150
REPO_151	Inventory Activity Detail	/app/reporting/reportrunner.nl?cr=151
REPO_152	Realized Exchange Rate Gains and Losses	/app/reporting/reportrunner.nl?cr=152
REPO_16	ALT_SALES Forecast vs. Quota	/app/reporting/reportrunner.nl?cr=16
REPO_161	Customers by Partner Detail	/app/reporting/reportrunner.nl?cr=161
REPO_162	Customers by Partner	/app/reporting/reportrunner.nl?cr=162
REPO_163	Closed Cases Snapshot	/app/reporting/reportrunner.nl?cr=163

Task ID	Page Label in NetSuite	URL
REPO_164	New Cases Snapshot	/app/reporting/reportrunner.nl?cr=164
REPO_165	Open Cases Snapshot	/app/reporting/reportrunner.nl?cr=165
REPO_169	Closed Case Analysis Detail	/app/reporting/reportrunner.nl?cr=169
REPO_17	Gross Lead Source Analysis Detail	/app/reporting/reportrunner.nl?cr=17
REPO_170	Closed Case Analysis	/app/reporting/reportrunner.nl?cr=170
REPO_171	Open Case Analysis Detail	/app/reporting/reportrunner.nl?open=T&cr=171
REPO_172	Open Case Analysis	/app/reporting/reportrunner.nl?open=T&cr=172
REPO_173	Customers by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=173
REPO_174	Customers by Sales Rep	/app/reporting/reportrunner.nl?cr=174
REPO_175	Customers by Territory Detail	/app/reporting/reportrunner.nl?cr=175
REPO_176	Customers by Territory	/app/reporting/reportrunner.nl?cr=176
REPO_177	Lead Source Analysis Detail	/app/reporting/reportrunner.nl?cr=177
REPO_178	Lead Source Analysis	/app/reporting/reportrunner.nl?cr=178
REPO_179	Sales Orders Register	/app/reporting/reportrunner.nl?cr=179
REPO_18	Gross Lead Source Analysis	/app/reporting/reportrunner.nl?cr=18
REPO_180	Open Sales Orders	/app/reporting/reportrunner.nl?cr=180
REPO_183	Shopping Cart Abandonment	/app/reporting/reportrunner.nl?cr=183
REPO_184	Shopping Activity Analysis by Category	/app/reporting/reportrunner.nl?cr=184
REPO_185	Shopping Activity Analysis	/app/reporting/reportrunner.nl?cr=185
REPO_188	Item Page Views and Sales Summary	/app/reporting/reportrunner.nl?cr=188
REPO_189	Hosted Page Hits Detail	/app/reporting/reportrunner.nl?cr=189
REPO_19	A/P Payment History by Payment	/app/reporting/reportrunner.nl?cr=19
REPO_191	Hosted Page Hits	/app/reporting/reportrunner.nl?cr=191
REPO_192	Item Orders Detail	/app/reporting/reportrunner.nl?cr=192
REPO_193	Item Orders by Category	/app/reporting/reportrunner.nl?cr=193
REPO_194	Item Orders	/app/reporting/reportrunner.nl?cr=194
REPO_195	Page Hits Detail	/app/reporting/reportrunner.nl?cr=195
REPO_196	Page Hits by Category	/app/reporting/reportrunner.nl?cr=196
REPO_197	Page Hits	/app/reporting/reportrunner.nl?cr=197
REPO_198	Sales by Promotion Detail	/app/reporting/reportrunner.nl?subtype=r&cr=198

Task ID	Page Label in NetSuite	URL
REPO_199	Sales by Promotion	/app/reporting/reportrunner.nl?subtype=r&cr=199
REPO_2	Deferred / Capitalized Expense	/app/reporting/reportrunner.nl?cr=2
REPO_20	A/P Payment History by Bill	/app/reporting/reportrunner.nl?cr=20
REPO_203	Prospect Analysis	/app/reporting/reportrunner.nl?cr=203
REPO_204	Prospect Analysis Detail	/app/reporting/reportrunner.nl?cr=204
REPO_205	Partner Commissions Pending Authorization	/app/reporting/reportrunner.nl?cr=205
REPO_206	Partner Commissions Pending Authorization Detail	/app/reporting/reportrunner.nl?cr=206
REPO_209	Unbilled Cost by Customer Detail	/app/reporting/reportrunner.nl?cr=209
REPO_21	A/R Payment History by Payment	/app/reporting/reportrunner.nl?cr=21
REPO_210	Unbilled Time by Customer Detail	/app/reporting/reportrunner.nl?cr=210
REPO_211	Unbilled Time by Customer	/app/reporting/reportrunner.nl?cr=211
REPO_214	Sales Tax Transaction Detail	/app/reporting/reportrunner.nl?cr=214
REPO_215	Sales Tax Liability By Tax Item	/app/reporting/reportrunner.nl?cr=215
REPO_218	Time by Item Detail	/app/reporting/reportrunner.nl?cr=218
REPO_219	Time by Customer Detail	/app/reporting/reportrunner.nl?cr=219
REPO_22	A/R Payment History by Invoice	/app/reporting/reportrunner.nl?cr=22
REPO_220	Time by Employee Detail	/app/reporting/reportrunner.nl?cr=220
REPO_221	Time by Item	/app/reporting/reportrunner.nl?cr=221
REPO_222	Time by Customer	/app/reporting/reportrunner.nl?cr=222
REPO_223	Time by Employee	/app/reporting/reportrunner.nl?cr=223
REPO_225	Unbilled Cost by Customer	/app/reporting/reportrunner.nl?cr=225
REPO_226	Inventory Revenue Detail	/app/reporting/reportrunner.nl?cr=226
REPO_227	Inventory Valuation Detail	/app/reporting/reportrunner.nl?cr=227
REPO_229	Inventory Revenue	/app/reporting/reportrunner.nl?cr=229
REPO_23	Comparative Sales (ALT_SALES)	/app/reporting/reportrunner.nl?cr=23
REPO_230	Inventory Valuation	/app/reporting/reportrunner.nl?cr=230
REPO_231	Inventory Profitability	/app/reporting/reportrunner.nl?cr=231
REPO_232	Payroll Liability	/app/reporting/reportrunner.nl?payitemtype=&cr=232
REPO_233	Payroll Journal	/app/reporting/reportrunner.nl?payitemtype=&cr=233
REPO_234	Payroll Summary by Employee	/app/reporting/reportrunner.nl?cr=-176

Task ID	Page Label in NetSuite	URL
REPO_235	Payroll Summary	/app/reporting/reportrunner.nl?payitemtype=&cr=235
REPO_239	Item Transaction Detail	/app/reporting/reportrunner.nl?cr=239
REPO_24	Comparative Sales (Orders)	/app/reporting/reportrunner.nl?cr=24
REPO_240	Sales Back Order Report	/app/reporting/reportrunner.nl?cr=240
REPO_241	Physical Inventory Worksheet	/app/reporting/reportrunner.nl?cr=241
REPO_242	Payroll Check Register	/app/reporting/reportrunner.nl?cr=242
REPO_243	Current Inventory Status	/app/reporting/reportrunner.nl?cr=243
REPO_246	Items Pending Fulfillment	/app/reporting/reportrunner.nl?cr=246
REPO_248	Cost by Customer Detail	/app/reporting/reportrunner.nl?cr=248
REPO_249	Estimated Partner Commission	/app/reporting/reportrunner.nl?cr=249
REPO_25	New Customer Sales Orders Detail	/app/reporting/reportrunner.nl?cr=25
REPO_250	Transaction Detail	/app/reporting/reportrunner.nl?cr=250
REPO_251	Account Detail	/app/reporting/reportrunner.nl?cr=251
REPO_256	Total Open Estimates	/app/reporting/reportrunner.nl?cr=256
REPO_257	Estimates Register	/app/reporting/reportrunner.nl?cr=257
REPO_258	Estimated Partner Commission Detail	/app/reporting/reportrunner.nl?cr=258
REPO_26	New Customer Sales Orders	/app/reporting/reportrunner.nl?cr=26
REPO_261	Open Invoices	/app/reporting/reportrunner.nl?cr=261
REPO_262	A/R Register	/app/reporting/reportrunner.nl?cr=262
REPO_264	Customer Profitability Detail	/app/reporting/reportrunner.nl?cr=264
REPO_266	Customer Profitability	/app/reporting/reportrunner.nl?cr=266
REPO_267	Sales by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=267
REPO_268	Sales by Item Detail	/app/reporting/reportrunner.nl?cr=268
REPO_269	Sales by Customer Detail	/app/reporting/reportrunner.nl?cr=269
REPO_27	Sales Orders by Promotion Detail	/app/reporting/reportrunner.nl?cr=27
REPO_270	Sales by Sales Rep	/app/reporting/reportrunner.nl?cr=270
REPO_271	Sales by Item	/app/reporting/reportrunner.nl?cr=271
REPO_272	Sales by Customer	/app/reporting/reportrunner.nl?cr=272
REPO_273	A/R Aging Detail	/app/reporting/reportrunner.nl?cr=273
REPO_274	A/R Aging	/app/reporting/reportrunner.nl?cr=274
REPO_277	Open Purchase Orders	/app/reporting/reportrunner.nl?viewasreport=T&specacct=PurchOrd&openonly=T&acctid=&cr=277



Task ID	Page Label in NetSuite	URL
REPO_278	Purchase Order Register	/app/reporting/reportrunner.nl?viewasreport=T&specacct=PurchOrd&reporttype=REGISTER&acctid=&cr=278
REPO_279	Purchase by Item Detail	/app/reporting/reportrunner.nl?subtype=i&cr=279
REPO_28	Sales Orders by Promotion	/app/reporting/reportrunner.nl?cr=28
REPO_280	Purchase by Vendor Detail	/app/reporting/reportrunner.nl?rtype=v&cr=280
REPO_281	Purchase by Item	/app/reporting/reportrunner.nl?subtype=i&cr=281
REPO_282	Purchase by Vendor	/app/reporting/reportrunner.nl?subtype=v&cr=282
REPO_283	Open Bills	/app/reporting/reportrunner.nl?viewasreport=T&specacct=AcctPay&openonly=T&acctid=&cr=283
REPO_284	A/P Register	/app/reporting/reportrunner.nl?viewasreport=T&specacct=AcctPay&reporttype=REGISTER&acctid=&cr=284
REPO_285	A/P Aging Detail	/app/reporting/reportrunner.nl?id=&ddue1=&ddue2=&cr=285
REPO_286	A/P Aging	/app/reporting/reportrunner.nl?cr=286
REPO_29	Sales Orders by Partner Detail	/app/reporting/reportrunner.nl?cr=29
REPO_292	Trial Balance	/app/reporting/reportrunner.nl?cr=292
REPO_293	General Ledger	/app/reporting/reportrunner.nl?cr=293
REPO_3	Amortization Forecast Detail	/app/reporting/reportrunner.nl?cr=3
REPO_30	Sales Orders by Partner	/app/reporting/reportrunner.nl?cr=30
REPO_309	Sales by Customer Detail filter for inv	/app/reporting/reportrunner.nl?cr=309&reload=T
REPO_31	Sales Orders by Historical Team Detail	/app/reporting/reportrunner.nl?cr=31
REPO_310	Back Ordered Items	/app/reporting/reportrunner.nl?cr=310&reload=T
REPO_318	Total Open Opportunites By Competitor	/app/reporting/reportrunner.nl?cr=318&reload=T
REPO_319	My Open Case Analysis Summary	/app/reporting/reportrunner.nl?cr=319&reload=T
REPO_32	Sales Orders by Historical Team	/app/reporting/reportrunner.nl?cr=32
REPO_325	Opportunities Lost By Competitor	/app/reporting/reportrunner.nl?cr=325&reload=T
REPO_326	Keyword - Customers by Keyword Family	/app/reporting/reportrunner.nl?cr=326&reload=T

Task ID	Page Label in NetSuite	URL
REPO_327	Keyword - Leads by Search Engine	/app/reporting/reportrunner.nl?cr=327&reload=T
REPO_327	0403 BS - Edit Text Row	/app/reporting/reportrunner.nl?cr=327&reload=T
REPO_328	Orders by Items by Class	/app/reporting/reportrunner.nl?cr=328&reload=T
REPO_329	Orders by Customer by Class	/app/reporting/reportrunner.nl?cr=329&reload=T
REPO_33	Sales Orders by Item Detail	/app/reporting/reportrunner.nl?cr=33
REPO_335	Email Campaign Statistics Summary	/app/reporting/reportrunner.nl?cr=335&reload=T
REPO_352	Utilization by Employee Report	/app/reporting/reportrunner.nl?cr=352&reload=T
REPO_353	Monthly Utilization by Consultant	/app/reporting/reportrunner.nl?cr=353&reload=T
REPO_354	Campaign ROI Detail	/app/reporting/reportrunner.nl?cr=354&reload=T
REPO_355	Campaign ROI Summary	/app/reporting/reportrunner.nl?cr=355&reload=T
REPO_36	Sales Orders by Sales Rep	/app/reporting/reportrunner.nl?cr=36
REPO_38	Sales Orders by Customer	/app/reporting/reportrunner.nl?cr=38
REPO_4	Amortization Forecast	/app/reporting/reportrunner.nl?cr=4
REPO_40	Integration and Automation Usage Detail By Job	/app/reporting/reportrunner.nl?cr=40
REPO_41	Integration and Automation Usage Summary By Job	/app/reporting/reportrunner.nl?cr=41
REPO_42	Integration and Automation Usage Summary By Record Type	/app/reporting/reportrunner.nl?cr=42
REPO_43	Hosted Page Hits by Customer	/app/reporting/reportrunner.nl?cr=43
REPO_44	Page Hits by Customers	/app/reporting/reportrunner.nl?cr=44
REPO_48	Forecast by Item Detail	/app/reporting/reportrunner.nl?cr=48
REPO_49	Forecast by Item	/app/reporting/reportrunner.nl?cr=49
REPO_5	Aggregated Sales Orders by Sales Rep Summary	/app/reporting/reportrunner.nl?cr=5
REPO_50	Open Escalations	/app/reporting/reportrunner.nl?cr=50
REPO_51	Total Pipeline by Statuses Summary	/app/reporting/reportrunner.nl?cr=51
REPO_53	Calculated Forecast by Sales Rep	/app/reporting/reportrunner.nl?cr=53
REPO_54	Visitor Activity	/app/reporting/reportrunner.nl?cr=54

Task ID	Page Label in NetSuite	URL
REPO_55	Lead Conversion	/app/reporting/reportrunner.nl?cr=55
REPO_56	Visitor Activity Detail	/app/reporting/reportrunner.nl?cr=56
REPO_57	Forecast by Sales Rep	/app/reporting/reportrunner.nl?cr=57
REPO_58	Sales Tax Liability By Tax Agency	/app/reporting/reportrunner.nl?cr=58
REPO_59	GST/HST on Purchases	/app/reporting/reportrunner.nl?cr=59
REPO_6	Authorized Partner Commission	/app/reporting/reportrunner.nl?cr=6
REPO_60	GST/HST on Sales	/app/reporting/reportrunner.nl?cr=60
REPO_61	GST/HST on Purchases Detail	/app/reporting/reportrunner.nl?cr=61
REPO_62	GST/HST on Sales Detail	/app/reporting/reportrunner.nl?cr=62
REPO_63	GST/HST Audit Summary	/app/reporting/reportrunner.nl?cr=63
REPO_64	Inventory Turnover	/app/reporting/reportrunner.nl?cr=64
REPO_65	Scheduled Deferred Revenue	/app/reporting/reportrunner.nl?cr=65
REPO_66	Revenue Recognition Forecast Detail	/app/reporting/reportrunner.nl?cr=66
REPO_67	Revenue Recognition Forecast	/app/reporting/reportrunner.nl?cr=67
REPO_68	Total Open Opportunities	/app/reporting/reportrunner.nl?cr=68
REPO_69	Opportunities to Close	/app/reporting/reportrunner.nl?cr=69
REPO_7	Authorized Partner Commission Detail	/app/reporting/reportrunner.nl?cr=7
REPO_71	Sales by Historical Team Detail	/app/reporting/reportrunner.nl?cr=71
REPO_72	Sales by Historical Team	/app/reporting/reportrunner.nl?cr=72
REPO_73	Forecast (Outstanding) by Customer Detail	/app/reporting/reportrunner.nl?cr=73
REPO_74	Forecast (Outstanding) by Customer	/app/reporting/reportrunner.nl?cr=74
REPO_77	State Withholding Detail	/app/reporting/reportrunner.nl?cr=77
REPO_78	State Withholding	/app/reporting/reportrunner.nl?cr=78
REPO_79	Hours & Earnings	/app/reporting/reportrunner.nl?cr=79
REPO_8	ALT_SALES Total Pipeline by Sales Rep Detail	/app/reporting/reportrunner.nl?cr=8
REPO_80	Closed Case Escalation	/app/reporting/reportrunner.nl?cr=80
REPO_81	Open Case Escalation	/app/reporting/reportrunner.nl?cr=81
REPO_82	Closed Case Escalation Detail	/app/reporting/reportrunner.nl?cr=82
REPO_83	Open Case Escalation Detail	/app/reporting/reportrunner.nl?cr=83
REPO_84	Payroll Detail	/app/reporting/reportrunner.nl?cr=-177
REPO_85	New Visitor	/app/reporting/reportrunner.nl?cr=85



Task ID	Page Label in NetSuite	URL
REPO_87	Keywords Detail	/app/reporting/reportrunner.nl?cr=87
REPO_88	Referrer Detail	/app/reporting/reportrunner.nl?cr=88
REPO_9	ALT_SALES Total Pipeline by Sales Rep Summary	/app/reporting/reportrunner.nl?cr=9
REPO_90	Keywords	/app/reporting/reportrunner.nl?cr=90
REPO_91	Referrer	/app/reporting/reportrunner.nl?cr=91
REPO_93	Payroll Liability Detail	/app/reporting/reportrunner.nl?cr=93
REPO_94	Total Pipeline by Customer	/app/reporting/reportrunner.nl?cr=94
REPO_95	Total Pipeline by Customer Detail	/app/reporting/reportrunner.nl?cr=95
REPO_96	Estimates to Close	/app/reporting/reportrunner.nl?cr=96
REPO_97	Opportunities to Close Detail	/app/reporting/reportrunner.nl?cr=97
REPO_98	Forecast by Customer Detail	/app/reporting/reportrunner.nl?cr=98
REPO_99	Forecast by Customer	/app/reporting/reportrunner.nl?cr=99
REPO_ASSIGNFINANCIALLA YOUTS	Row Layout Assignment	/app/reporting/ financiallayoutassignments.nl
REPO_BANKREGISTER	Bank Register	/app/reporting/reportrunner.nl? reporttype=REGISTER&accttype=Bank& code=CHECK_REG
REPO_CUSTOMIZATION	Generic Report Customization	/app/reporting/reportcomposer.nl
REPO_CUST_10	Customize ALT_SALES Pipeline by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=10&e=T
REPO_CUST_100	Customize Total Pipeline by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=100&e=T
REPO_CUST_101	Customize Total Pipeline by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=101&e=T
REPO_CUST_102	Customize New Customer Sales Detail	/app/reporting/reportcomposer.nl?cr=102&e=T
REPO_CUST_103	Customize New Customer Sales Summary	/app/reporting/reportcomposer.nl?cr=103&e=T
REPO_CUST_104	Customize Opportunities Lost	/app/reporting/reportcomposer.nl?cr=104&e=T
REPO_CUST_105	Customize Opportunities Won	/app/reporting/reportcomposer.nl?cr=105&e=T
REPO_CUST_106	Customize Sales Activity by Customer Summary	/app/reporting/reportcomposer.nl?cr=106&e=T
REPO_CUST_107	Customize Sales Activity by Customer Detail	/app/reporting/reportcomposer.nl?cr=107&e=T
REPO_CUST_11	Customize ALT_SALES Pipeline by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=11&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_111	Customize Authorized Commissions Summary	/app/reporting/reportcomposer.nl?cr=111&e=T
REPO_CUST_112	Customize Authorized Commissions Detail	/app/reporting/reportcomposer.nl?cr=112&e=T
REPO_CUST_113	Customize GST on Purchases Summary	/app/reporting/reportcomposer.nl?cr=113&e=T
REPO_CUST_114	Customize GST on Sales Summary	/app/reporting/reportcomposer.nl?cr=114&e=T
REPO_CUST_115	Customize GST on Purchases Detail	/app/reporting/reportcomposer.nl?cr=115&e=T
REPO_CUST_116	Customize GST on Sales Detail	/app/reporting/reportcomposer.nl?cr=116&e=T
REPO_CUST_118	Customize Commissions Pending Authorization Summary	/app/reporting/reportcomposer.nl?cr=118&e=T
REPO_CUST_119	Customize Commissions Pending Authorization Detail	/app/reporting/reportcomposer.nl?cr=119&e=T
REPO_CUST_12	Customize ALT_SALES Forecast by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=12&e=T
REPO_CUST_121	Customize Estimated Commissions Summary	/app/reporting/reportcomposer.nl?cr=121&e=T
REPO_CUST_122	Customize Estimated Commissions Detail	/app/reporting/reportcomposer.nl?cr=122&e=T
REPO_CUST_123	Customize Campaign Response Detail	/app/reporting/reportcomposer.nl?cr=123&e=T
REPO_CUST_126	Customize Pipeline by Customer Summary	/app/reporting/reportcomposer.nl?cr=126&e=T
REPO_CUST_127	Customize Pipeline by Customer Detail	/app/reporting/reportcomposer.nl?cr=127&e=T
REPO_CUST_128	Customize Pipeline by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=128&e=T
REPO_CUST_129	Customize Pipeline ROI by Manager	/app/reporting/reportcomposer.nl?cr=129&e=T
REPO_CUST_13	Customize ALT_SALES Forecast by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=13&e=T
REPO_CUST_130	Customize Pipeline by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=130&e=T
REPO_CUST_131	Customize Pipeline by Status Summary	/app/reporting/reportcomposer.nl?cr=131&e=T
REPO_CUST_132	Customize Campaign ROI Analysis Detail	/app/reporting/reportcomposer.nl?cr=132&e=T
REPO_CUST_133	Customize Campaign Response Summary	/app/reporting/reportcomposer.nl?cr=133&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_134	Customize Sales Activity Summary	/app/reporting/reportcomposer.nl?cr=134&e=T
REPO_CUST_135	Customize Sales Activity Detail	/app/reporting/reportcomposer.nl?cr=135&e=T
REPO_CUST_137	Customize VAT on Purchases Summary	/app/reporting/reportcomposer.nl?cr=137&e=T
REPO_CUST_138	Customize VAT on Sales Summary	/app/reporting/reportcomposer.nl?cr=138&e=T
REPO_CUST_139	Customize VAT on Purchases Detail	/app/reporting/reportcomposer.nl?cr=139&e=T
REPO_CUST_14	Customize Billings Forecast vs. Quota Report	/app/reporting/reportcomposer.nl?cr=14&e=T
REPO_CUST_140	Customize VAT on Sales Detail	/app/reporting/reportcomposer.nl?cr=140&e=T
REPO_CUST_141	Customize Partner Activity Summary	/app/reporting/reportcomposer.nl?cr=141&e=T
REPO_CUST_142	Customize Partner Activity Detail	/app/reporting/reportcomposer.nl?cr=142&e=T
REPO_CUST_143	Customize Sales by Partner Summary	/app/reporting/reportcomposer.nl?cr=143&e=T
REPO_CUST_144	Customize Sales by Partner Detail	/app/reporting/reportcomposer.nl?cr=144&e=T
REPO_CUST_145	Customize Shipping Report	/app/reporting/reportcomposer.nl?cr=145&e=T
REPO_CUST_146	Customize Total Open Opportunities Detail	/app/reporting/reportcomposer.nl?cr=146&e=T
REPO_CUST_147	Customize Forecast vs. Quota	/app/reporting/reportcomposer.nl?cr=147&e=T
REPO_CUST_148	Customize Sales Orders Pending Fulfillment	/app/reporting/reportcomposer.nl?cr=148&e=T
REPO_CUST_149	Customize Forecast Detail by Sales Rep	/app/reporting/reportcomposer.nl?cr=149&e=T
REPO_CUST_15	Customize Bookings and Billings Forecast vs. Quota Report	/app/reporting/reportcomposer.nl?cr=15&e=T
REPO_CUST_150	Customize Inventory Back Order Report	/app/reporting/reportcomposer.nl?cr=150&e=T
REPO_CUST_151	Customize Inventory Activity Detail	/app/reporting/reportcomposer.nl?cr=151&e=T
REPO_CUST_16	Customize Bookings Forecast vs. Quota Report	/app/reporting/reportcomposer.nl?cr=16&e=T
REPO_CUST_161	Customize Customers by Partner Detail	/app/reporting/reportcomposer.nl?cr=161&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_162	Customize Customers by Partner Summary	/app/reporting/reportcomposer.nl?cr=162&e=T
REPO_CUST_169	Customize Closed Case Analysis Detail	/app/reporting/reportcomposer.nl?cr=169&e=T
REPO_CUST_17	Customize Gross Lead Analysis Detail Report	/app/reporting/reportcomposer.nl?cr=17&e=T
REPO_CUST_170	Customize Closed Case Analysis Summary	/app/reporting/reportcomposer.nl?cr=170&e=T
REPO_CUST_171	Customize Open Case Analysis Detail	/app/reporting/reportcomposer.nl?cr=171&e=T&open=T
REPO_CUST_172	Customize Open Case Analysis Summary	/app/reporting/reportcomposer.nl?cr=172&e=T&open=T
REPO_CUST_173	Customize Customers by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=173&e=T
REPO_CUST_174	Customize Customers by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=174&e=T
REPO_CUST_175	Customize Customers by Territory Detail	/app/reporting/reportcomposer.nl?cr=175&e=T
REPO_CUST_176	Customize Customers by Territory Summary	/app/reporting/reportcomposer.nl?cr=176&e=T
REPO_CUST_177	Customize Lead Source Analysis Detail	/app/reporting/reportcomposer.nl?cr=177&e=T
REPO_CUST_178	Customize Lead Source Analysis Summary	/app/reporting/reportcomposer.nl?cr=178&e=T
REPO_CUST_179	Customize Sales Orders Register	/app/reporting/reportcomposer.nl?cr=179&e=T
REPO_CUST_18	Customize Gross Lead Analysis Summary Report	/app/reporting/reportcomposer.nl?cr=18&e=T
REPO_CUST_180	Customize Open Sales Orders	/app/reporting/reportcomposer.nl?cr=180&e=T
REPO_CUST_183	Customize Shopping Activity Analysis by Visitor	/app/reporting/reportcomposer.nl?cr=183&e=T
REPO_CUST_184	Customize Shopping Activity Analysis by Category	/app/reporting/reportcomposer.nl?cr=184
REPO_CUST_185	Customize Shopping Activity Analysis	/app/reporting/reportcomposer.nl?e=T&cr=185
REPO_CUST_188	Customize Item Page Views and Sales Summary	/app/reporting/reportcomposer.nl?e=T&cr=188
REPO_CUST_189	Customize Hosted Page Hits by Visitor	/app/reporting/reportcomposer.nl?cr=189&e=T
REPO_CUST_19	A/P Payment History by Payment	/app/reporting/reportcomposer.nl?cr=19&e=T



Task ID	Page Label in NetSuite	URL
REPO_CUST_191	Customize Hosted Page Hits	/app/reporting/reportcomposer.nl?cr=191&e=T
REPO_CUST_192	Customize Item Order History Detail	/app/reporting/reportcomposer.nl?cr=192&e=T
REPO_CUST_193	Customize Item Order History by Category	/app/reporting/reportcomposer.nl?cr=193
REPO_CUST_194	Customize Item Order History	/app/reporting/reportcomposer.nl?e=T&cr=194
REPO_CUST_195	Customize Page Hits Detail	/app/reporting/reportcomposer.nl?cr=195&e=T
REPO_CUST_196	Customize Page Hits by Category	/app/reporting/reportcomposer.nl?cr=196&e=T
REPO_CUST_197	Customize Page Hits	/app/reporting/reportcomposer.nl?e=T&cr=197
REPO_CUST_198	Customize Sales by Promotion Detail	/app/reporting/reportcomposer.nl?e=T&payitemtype=r&cr=198
REPO_CUST_199	Customize Sales by Promotion Summary	/app/reporting/reportcomposer.nl?e=T&subtype=r&cr=199
REPO_CUST_2	Customize Deferred / Capitalized Expense	/app/reporting/reportcomposer.nl?cr=2&e=T
REPO_CUST_20	A/P Payment History by Bill	/app/reporting/reportcomposer.nl?cr=20&e=T
REPO_CUST_203	Customize Prospect Analysis Summary	/app/reporting/reportcomposer.nl?e=T&cr=203
REPO_CUST_204	Customize Prospect Analysis Detail	/app/reporting/reportcomposer.nl?e=T&cr=204
REPO_CUST_205	Customize Partner Commission Pending Authorization Summary	/app/reporting/reportcomposer.nl?e=T&cr=205
REPO_CUST_206	Customize Partner Commission Pending Authorization Detail	/app/reporting/reportcomposer.nl?e=T&cr=206
REPO_CUST_209	Customize Unbilled Cost by Customer Detail	/app/reporting/reportcomposer.nl?e=T&cr=209
REPO_CUST_21	A/R Payment History by Payment	/app/reporting/reportcomposer.nl?cr=21&e=T
REPO_CUST_210	Customize Unbilled Time by Customer Detail	/app/reporting/reportcomposer.nl?e=T&cr=210
REPO_CUST_211	Customize Unbilled Time by Customer Summary	/app/reporting/reportcomposer.nl?e=T&cr=211
REPO_CUST_214	Customize Sales Tax Transaction Detail	/app/reporting/reportcomposer.nl?cr=214&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_215	Customize Sales Tax Liability By Tax Item	/app/reporting/reportcomposer.nl?cr=215&e=T
REPO_CUST_218	Customize Time by Item Detail	/app/reporting/reportcomposer.nl?cr=218&e=T
REPO_CUST_219	Customize Time by Customer Detail	/app/reporting/reportcomposer.nl?cr=219&e=T
REPO_CUST_22	A/R Payment History by Invoice	/app/reporting/reportcomposer.nl?cr=22&e=T
REPO_CUST_220	Customize Time by Employee Detail	/app/reporting/reportcomposer.nl?cr=220&e=T
REPO_CUST_221	Customize Time by Item Summary	/app/reporting/reportcomposer.nl?cr=221&e=T
REPO_CUST_222	Customize Time by Customer Summary	/app/reporting/reportcomposer.nl?cr=222&e=T
REPO_CUST_223	Customize Time by Employee Summary	/app/reporting/reportcomposer.nl?cr=223&e=T
REPO_CUST_225	Customize Unbilled Cost by Customer Summary	/app/reporting/reportcomposer.nl?e=T&cr=225
REPO_CUST_226	Customize Inventory Revenue Detail	/app/reporting/reportcomposer.nl?cr=226&e=T
REPO_CUST_227	Customize Inventory Valuation Detail	/app/reporting/reportcomposer.nl?e=T&cr=227
REPO_CUST_229	Customize Inventory Revenue Summary	/app/reporting/reportcomposer.nl?cr=229&e=T
REPO_CUST_230	Customize Inventory Valuation Summary	/app/reporting/reportcomposer.nl?cr=230&e=T
REPO_CUST_231	Customize Inventory Profitability	/app/reporting/reportcomposer.nl?cr=231&e=T
REPO_CUST_232	Customize Payroll Liabilities	/app/reporting/reportrunner.nl?payitemtype=&cr=232
REPO_CUST_233	Customize Payroll Journal	/app/reporting/reportrunner.nl?payitemtype=&cr=233
REPO_CUST_234	Customize Payroll Summary by Employee	/app/reporting/reportrunner.nl?cr=-176
REPO_CUST_235	Customize Payroll Summary by Item	/app/reporting/reportrunner.nl?payitemtype=&cr=235
REPO_CUST_239	Customize Item Transaction Detail	/app/reporting/reportcomposer.nl?cr=239&e=T
REPO_CUST_240	Customize Sales Back Order Report	/app/reporting/reportcomposer.nl?cr=240&e=T
REPO_CUST_241	Customize Physical Inventory Worksheet	/app/reporting/reportcomposer.nl?cr=241&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_242	Customize Payroll Check Register	/app/reporting/reportrunner.nl?cr=242
REPO_CUST_243	Customize Current Inventory Status	/app/reporting/reportcomposer.nl?cr=243&e=T
REPO_CUST_246	Customize Items Pending Fulfillment	/app/reporting/reportcomposer.nl?e=T&viewasreport=T&specacct=SalesOrd&reporttype=REGISTER&code=OPEN_SALES_ORDERS_DETAIL&acctid=&cr=246
REPO_CUST_248	Customize Cost by Customer Detail	/app/reporting/reportcomposer.nl?cr=248&e=T
REPO_CUST_249	Customize Estimated Partner Commission Summary	/app/reporting/reportcomposer.nl?e=T&cr=249
REPO_CUST_25	Customize New Customer Sales Orders Detail	/app/reporting/reportcomposer.nl?cr=25&e=T
REPO_CUST_250	Customize Transaction Detail	/app/reporting/reportcomposer.nl?e=T&cr=250
REPO_CUST_251	Customize Account Detail	/app/reporting/reportcomposer.nl?cr=251&e=T
REPO_CUST_256	Customize Total Open Estimates	/app/reporting/reportcomposer.nl?cr=256&e=T
REPO_CUST_257	Customize Estimates Register	/app/reporting/reportcomposer.nl?cr=257&e=T
REPO_CUST_258	Customize Estimated Partner Commission Detail	/app/reporting/reportcomposer.nl?e=T&cr=258
REPO_CUST_26	Customize New Customer Sales Orders Summary	/app/reporting/reportcomposer.nl?cr=26&e=T
REPO_CUST_261	Customize Open Invoices	/app/reporting/reportcomposer.nl?cr=261&e=T
REPO_CUST_262	Customize A/R Register	/app/reporting/reportcomposer.nl?cr=262&e=T
REPO_CUST_264	Customize Customer Profitability Detail	/app/reporting/reportcomposer.nl?e=T&cr=264
REPO_CUST_266	Customize Customer Profitability Summary	/app/reporting/reportcomposer.nl?cr=266&e=T
REPO_CUST_267	Customize Sales by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=267&e=T
REPO_CUST_268	Customize Sales by Item Detail	/app/reporting/reportcomposer.nl?cr=268&e=T
REPO_CUST_269	Customize Sales by Customer Detail	/app/reporting/reportcomposer.nl?cr=269&e=T
REPO_CUST_27	Customize Sales Orders by Promotion Detail	/app/reporting/reportcomposer.nl?cr=27&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_270	Customize Sales by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=270&e=T
REPO_CUST_271	Customize Sales by Item Summary	/app/reporting/reportcomposer.nl?cr=271&e=T
REPO_CUST_272	Customize Sales by Customer Summary	/app/reporting/reportcomposer.nl?cr=272&e=T
REPO_CUST_273	Customize A/R Aging Detail	/app/reporting/reportcomposer.nl?cr=273&e=T
REPO_CUST_274	Customize A/R Aging Summary	/app/reporting/reportcomposer.nl?cr=274&e=T
REPO_CUST_277	Customize Open Purchase Orders	/app/reporting/reportcomposer.nl?e=T&viewasreport=T&specacct=PurchOrd&openonly=T&reporttype=REGISTER&code=OPEN_PO&acctid=&cr=277
REPO_CUST_278	Customize Purchase Order Register	/app/reporting/reportcomposer.nl?e=T&viewasreport=T&specacct=PurchOrd&reporttype=REGISTER&code=PO_REG&acctid=&cr=278
REPO_CUST_279	Customize Purchase by Item Detail	/app/reporting/reportcomposer.nl?e=T&payitemtype=i&code=PURCH_DETAIL_BY_ITEM&cr=279
REPO_CUST_28	Customize Sales Orders by Promotion Summary	/app/reporting/reportcomposer.nl?cr=28&e=T
REPO_CUST_280	Customize Purchase by Vendor Detail	/app/reporting/reportcomposer.nl?e=T&rtype=v&code=PURCH_DETAIL_BY_VENDOR&cr=280
REPO_CUST_281	Customize Purchase by Item Summary	/app/reporting/reportcomposer.nl?e=T&payitemtype=i&code=PURCH_SUM_BY_ITEM&cr=281
REPO_CUST_282	Customize Purchase by Vendor Summary	/app/reporting/reportcomposer.nl?e=T&payitemtype=v&code=PURCH_SUM_BY_VENDOR&cr=282
REPO_CUST_283	Customize Open Bills	/app/reporting/reportcomposer.nl?e=T&viewasreport=T&specacct=AcctPay&openonly=T&reporttype=REGISTER&code=OPEN_BILLS&acctid=&cr=283
REPO_CUST_284	Customize A/P Register	/app/reporting/reportcomposer.nl?e=T&viewasreport=T&specacct=AcctPay&reporttype=REGISTER&code=AP_REG&acctid=&cr=284
REPO_CUST_285	Customize A/P Aging Detail	/app/reporting/reportcomposer.nl?e=T&acctid=&ddue1=&ddue2=&code=AP_AGING_DETAIL&cr=285
REPO_CUST_286	Customize A/P Aging Summary	/app/reporting/reportcomposer.nl?e=T&cr=286

Task ID	Page Label in NetSuite	URL
REPO_CUST_29	Customize Sales Orders by Partner Detail	/app/reporting/reportcomposer.nl?cr=29&e=T
REPO_CUST_292	Customize Trial Balance	/app/reporting/reportcomposer.nl?e=T&cr=292
REPO_CUST_293	Customize General Ledger	/app/reporting/reportcomposer.nl?e=T&cr=293
REPO_CUST_3	Customize Amortization Forecast Detail	/app/reporting/reportcomposer.nl?cr=3&e=T
REPO_CUST_30	Customize Sales Orders by Partner Summary	/app/reporting/reportcomposer.nl?cr=30&e=T
REPO_CUST_31	Customize Sales Orders by Historical Team Detail	/app/reporting/reportcomposer.nl?cr=31&e=T
REPO_CUST_310	Customize Back Ordered Items	/app/reporting/reportcomposer.nl?cr=310
REPO_CUST_33	Customize Sales Orders by Item Detail	/app/reporting/reportcomposer.nl?cr=33&e=T
REPO_CUST_34	Customize Sales Orders by Item Summary	/app/reporting/reportcomposer.nl?cr=34&e=T
REPO_CUST_355	Customize Campaign ROI Summary	/app/reporting/reportcomposer.nl?e=T&cr=355
REPO_CUST_39	Customize Forecast by Status Summary	/app/reporting/reportcomposer.nl?cr=39&e=T
REPO_CUST_4	Customize Amortization Forecast Summary	/app/reporting/reportcomposer.nl?cr=4&e=T
REPO_CUST_40	Customize Integration and Automation Usage Detail By Job	/app/reporting/reportcomposer.nl?cr=40&e=T
REPO_CUST_41	Customize Integration and Automation Usage Summary By Job	/app/reporting/reportcomposer.nl?cr=41&e=T
REPO_CUST_42	Customize Integration and Automation Usage Summary By Record Type	/app/reporting/reportcomposer.nl?cr=42&e=T
REPO_CUST_43	Hosted Page Hits by Customer	/app/reporting/reportcomposer.nl?cr=43&e=T
REPO_CUST_44	Page Hits by Customer	/app/reporting/reportcomposer.nl?cr=44&e=T
REPO_CUST_48	Customize Forecast by Item Detail	/app/reporting/reportcomposer.nl?cr=48&e=T
REPO_CUST_49	Customize Forecast by Item Summary	/app/reporting/reportcomposer.nl?cr=49&e=T
REPO_CUST_5	Customize Aggregated Sales Orders by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=5&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_51	Customize Total Pipeline by Statuses	/app/reporting/reportcomposer.nl?cr=51&e=T
REPO_CUST_53	Customize Calculated Forecast by Sales Rep	/app/reporting/reportcomposer.nl?cr=53&e=T
REPO_CUST_54	Customize Visitor Activity Summary	/app/reporting/reportcomposer.nl?cr=54&e=T
REPO_CUST_55	Customize Lead Conversion	/app/reporting/reportcomposer.nl?cr=55&e=T
REPO_CUST_56	Customize Visitor Activity Detail	/app/reporting/reportcomposer.nl?cr=56&e=T
REPO_CUST_57	Customize Forecast by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=57&e=T
REPO_CUST_58	Customize Sales Tax Liability By Tax Agency	/app/reporting/reportcomposer.nl?cr=58&e=T
REPO_CUST_59	Customize GST/HST on Purchases Summary	/app/reporting/reportrunner.nl?cr=59&e=T
REPO_CUST_6	Customize Authorized Partner Commission Summary	/app/reporting/reportcomposer.nl?cr=6&e=T
REPO_CUST_60	Customize GST/HST on Sales Summary	/app/reporting/reportrunner.nl?cr=60&e=T
REPO_CUST_61	Customize GST/HST on Purchases Detail	/app/reporting/reportrunner.nl?cr=61&e=T
REPO_CUST_62	Customize GST/HST on Sales Detail	/app/reporting/reportrunner.nl?cr=62&e=T
REPO_CUST_63	Customize GST/HST Audit Summary	/app/reporting/reportrunner.nl?cr=63&e=T
REPO_CUST_64	Customize Inventory Turnover	/app/reporting/reportcomposer.nl?cr=64&e=T
REPO_CUST_65	Customize Deferred Revenue Forecast Summary	/app/reporting/reportcomposer.nl?cr=65&e=T
REPO_CUST_66	Customize Deferred Revenue Detail	/app/reporting/reportcomposer.nl?cr=66&e=T
REPO_CUST_67	Customize Deferred Revenue by Customer Summary	/app/reporting/reportcomposer.nl?cr=67&e=T
REPO_CUST_68	Customize Total Open Opportunities Summary	/app/reporting/reportcomposer.nl?cr=68&e=T
REPO_CUST_69	Customize Opportunities to Close Summary	/app/reporting/reportcomposer.nl?cr=69&e=T
REPO_CUST_7	Customize Authorized Partner Commission Detail	/app/reporting/reportcomposer.nl?cr=7&e=T
REPO_CUST_71	Customize Sales by Historical Team Detail	/app/reporting/reportcomposer.nl?cr=71&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_72	Customize Sales by Historical Team Summary	/app/reporting/reportcomposer.nl?cr=72&e=T
REPO_CUST_73	Customize Forecast (Outstanding) by Customer Detail	/app/reporting/reportcomposer.nl?cr=73&e=T
REPO_CUST_74	Customize Forecast (Outstanding) by Customer Summary	/app/reporting/reportcomposer.nl?cr=74&e=T
REPO_CUST_8	Customize ALT_SALES Total Pipeline by Sales Rep Detail	/app/reporting/reportcomposer.nl?cr=8&e=T
REPO_CUST_80	Closed Case Escalation Summary Customize	/app/reporting/reportcomposer.nl?cr=80&e=T
REPO_CUST_81	Open Case Escalation Summary Customize	/app/reporting/reportcomposer.nl?cr=81&e=T
REPO_CUST_82	Closed Case Escalation Detail Customize	/app/reporting/reportcomposer.nl?cr=82&e=T
REPO_CUST_83	Open Case Escalation Detail Customize	/app/reporting/reportcomposer.nl?cr=83&e=T
REPO_CUST_84	Customize Payroll Detail	/app/reporting/reportrunner.nl?cr=-177
REPO_CUST_85	New Visitor	/app/reporting/reportcomposer.nl?cr=85&e=T
REPO_CUST_87	Keywords Detail	/app/reporting/reportcomposer.nl?cr=87&e=T
REPO_CUST_88	Referrer Detail	/app/reporting/reportcomposer.nl?cr=88&e=T
REPO_CUST_9	Customize ALT_SALES Total Pipeline by Sales Rep Summary	/app/reporting/reportcomposer.nl?cr=9&e=T
REPO_CUST_90	Keywords Summary	/app/reporting/reportcomposer.nl?cr=90&e=T
REPO_CUST_91	Referrer Summary	/app/reporting/reportcomposer.nl?cr=91&e=T
REPO_CUST_94	Customize Total Pipeline by Customer Summary	/app/reporting/reportcomposer.nl?cr=94&e=T
REPO_CUST_95	Customize Total Pipeline by Customer Detail	/app/reporting/reportcomposer.nl?cr=95&e=T
REPO_CUST_96	Customize Estimates to Close	/app/reporting/reportcomposer.nl?cr=96&e=T
REPO_CUST_97	Customize Opportunities to Close Detail	/app/reporting/reportcomposer.nl?cr=97&e=T
REPO_CUST_98	Customize Forecast by Customer Detail	/app/reporting/reportcomposer.nl?cr=98&e=T
REPO_CUST_99	Customize Forecast by Customer Summary	/app/reporting/reportcomposer.nl?cr=99&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG100	Customize Case Activity by Support Rep Summary	/app/reporting/reportcomposer.nl?cr=100&e=T
REPO_CUST_NEG101	Customize Case Activity by Support Rep Detail	/app/reporting/reportcomposer.nl?cr=101&e=T
REPO_CUST_NEG105	Customize Utilization by Employee Summary (deprecated)	/app/reporting/reportcomposer.nl?cr=105&e=T
REPO_CUST_NEG106	Customize Current Backlog by Resource	/app/reporting/reportcomposer.nl?cr=106&e=T
REPO_CUST_NEG108	Customize Sales Orders by Sales Team Summary	/app/reporting/reportcomposer.nl?cr=108&e=T
REPO_CUST_NEG109	Customize Sales Orders by Sales Team Detail	/app/reporting/reportcomposer.nl?cr=109&e=T
REPO_CUST_NEG110	Customize Sales by Sales Team Summary	/app/reporting/reportcomposer.nl?cr=110&e=T
REPO_CUST_NEG111	Customize Sales by Sales Team Detail	/app/reporting/reportcomposer.nl?cr=111&e=T
REPO_CUST_NEG112	Customize Aggregated Sales Orders by Customer Summary	/app/reporting/reportcomposer.nl?cr=112&e=T
REPO_CUST_NEG113	Customize Aggregated Sales Orders by Item Summary	/app/reporting/reportcomposer.nl?cr=113&e=T
REPO_CUST_NEG114	Customize Internal Search Summary	/app/reporting/reportcomposer.nl?cr=114&e=T
REPO_CUST_NEG115	Customize Internal Search Detail	/app/reporting/reportcomposer.nl?cr=115&e=T
REPO_CUST_NEG116	Customize Customer Aging History	/app/reporting/reportcomposer.nl?cr=116&e=T
REPO_CUST_NEG118	Customize Estimated Profitability by Job	/app/reporting/reportcomposer.nl?cr=118&e=T
REPO_CUST_NEG120	Customize Sales Tax on Sales Summary	/app/reporting/reportcomposer.nl?cr=120&e=T
REPO_CUST_NEG121	Customize Sales Tax on Sales Detail	/app/reporting/reportcomposer.nl?cr=121&e=T
REPO_CUST_NEG122	Customize Campaign ROI Analysis Summary	/app/reporting/reportcomposer.nl?cr=122&e=T
REPO_CUST_NEG123	Customize Sales by Lead Source Detail	/app/reporting/reportcomposer.nl?cr=123&e=T
REPO_CUST_NEG124	Customize Sales by Lead Source Summary	/app/reporting/reportcomposer.nl?cr=124&e=T
REPO_CUST_NEG125	Customize PST on Sales Detail	/app/reporting/reportrunner.nl?cr=125&e=T
REPO_CUST_NEG126	Customize PST on Purchases Detail	/app/reporting/reportrunner.nl?cr=126&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG127	Customize PST on Sales Summary	/app/reporting/reportrunner.nl?cr=-127&e=T
REPO_CUST_NEG128	Customize PST on Purchases Summary	/app/reporting/reportrunner.nl?cr=-128&e=T
REPO_CUST_NEG130	Customize Bank Register	/app/reporting/reportcomposer.nl?cr=-130&e=T
REPO_CUST_NEG131	Customize Sales Tax Analysis	/app/reporting/reportcomposer.nl?cr=-131&e=T
REPO_CUST_NEG132	Customize Realized Exchange Rate Gains and Losses	/app/reporting/reportcomposer.nl?cr=-132&e=T
REPO_CUST_NEG133	Customize Unrealized Exchange Rate Gains and Losses	/app/reporting/reportcomposer.nl?cr=-133&e=T
REPO_CUST_NEG134	Customize Sales by Paid Keyword Detail	/app/reporting/reportcomposer.nl?cr=-134&e=T
REPO_CUST_NEG135	Customize Sales by Paid Keyword Summary	/app/reporting/reportcomposer.nl?cr=-135&e=T
REPO_CUST_NEG136	Customize Leads by Paid Keyword Detail	/app/reporting/reportcomposer.nl?cr=-136&e=T
REPO_CUST_NEG137	Customize Leads by Paid Keyword Summary	/app/reporting/reportcomposer.nl?cr=-137&e=T
REPO_CUST_NEG138	Customize Closed Issues Summary	/app/reporting/reportcomposer.nl?cr=-138&e=T&reload=T
REPO_CUST_NEG139	Customize Closed Issues Detail	/app/reporting/reportcomposer.nl?cr=-139&e=T&reload=T
REPO_CUST_NEG140	Customize Open Issues Summary	/app/reporting/reportcomposer.nl?cr=-140&e=T&reload=T
REPO_CUST_NEG141	Customize Open Issues Detail	/app/reporting/reportcomposer.nl?cr=-141&e=T&reload=T
REPO_CUST_NEG144	Customize Paid Employee Commission Summary	/app/reporting/reportcomposer.nl?cr=-144&e=T&reload=T
REPO_CUST_NEG145	Customize Paid Employee Commission Detail	/app/reporting/reportcomposer.nl?cr=-145&e=T&reload=T
REPO_CUST_NEG146	Customize Paid Partner Commission Summary	/app/reporting/reportcomposer.nl?cr=-146&e=T&reload=T
REPO_CUST_NEG147	Customize Paid Partner Commission Detail	/app/reporting/reportcomposer.nl?cr=-147&e=T&reload=T
REPO_CUST_NEG148	Customize Sales Order Revenue Forecast Summary	/app/reporting/reportcomposer.nl?cr=-148&e=T&reload=T
REPO_CUST_NEG149	Customize Sales Order Revenue Forecast Detail	/app/reporting/reportcomposer.nl?cr=-149&e=T&reload=T
REPO_CUST_NEG150	Customize Forecast vs. Quota (including Class)	/app/reporting/reportcomposer.nl?cr=-150&e=T&reload=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG151	Customize Forecast Accuracy	/app/reporting/reportcomposer.nl?cr=-151&e=T&reload=T
REPO_CUST_NEG152	Customize Forecast (ALT_SALES)	/app/reporting/reportcomposer.nl?cr=-152&e=T&reload=T
REPO_CUST_NEG153	Customize Calculated Forecast Accuracy	/app/reporting/reportcomposer.nl?cr=-153&e=T&reload=T
REPO_CUST_NEG154	Customize Calculated Forecast Accuracy (ALT_SALES)	/app/reporting/reportcomposer.nl?cr=-154&e=T&reload=T
REPO_CUST_NEG155	Customize Forecast vs. Quota (including Class)	/app/reporting/reportcomposer.nl?cr=-155&e=T&reload=T
REPO_CUST_NEG156	Customize Forecast vs. Quota (including Department)	/app/reporting/reportcomposer.nl?cr=-156&e=T&reload=T
REPO_CUST_NEG157	Customize ALT_SALES Forecast vs. Quota (including Class)	/app/reporting/reportcomposer.nl?cr=-157&e=T&reload=T
REPO_CUST_NEG158	Customize ALT_SALES Forecast vs. Quota (including Department)	/app/reporting/reportcomposer.nl?cr=-158&e=T&reload=T
REPO_CUST_NEG159	Customize ALT_SALES Forecast vs. Quota (including Location)	/app/reporting/reportcomposer.nl?cr=-159&e=T&reload=T
REPO_CUST_NEG160	Customize Forecast Accuracy	/app/reporting/reportcomposer.nl?cr=-160&e=T&reload=T
REPO_CUST_NEG161	Customize Calculated Forecast Accuracy	/app/reporting/reportcomposer.nl?cr=-161&e=T&reload=T
REPO_CUST_NEG162	Customize Earned Value by Job	/app/reporting/reportcomposer.nl?cr=-162&e=T
REPO_CUST_NEG175	Customize Service Fees	/app/reporting/reportrunner.nl?cr=-175
REPO_CUST_NEG178	Customize Utilization by Employee (deprecated)	/app/reporting/reportcomposer.nl?cr=-178&e=T
REPO_CUST_NEG179	Customize Deferred Revenue By Customer	/app/reporting/reportcomposer.nl?cr=-179&e=T
REPO_CUST_NEG180	Customize Deferred Revenue By Item	/app/reporting/reportcomposer.nl?cr=-180&e=T
REPO_CUST_NEG181	Customize Deferred Revenue By State	/app/reporting/reportcomposer.nl?cr=-181&e=T
REPO_CUST_NEG182	Customize Revenue By Customer	/app/reporting/reportcomposer.nl?cr=-182&e=T
REPO_CUST_NEG183	Customize Revenue By Item	/app/reporting/reportcomposer.nl?cr=-183&e=T
REPO_CUST_NEG184	Customize Revenue By State	/app/reporting/reportcomposer.nl?cr=-184&e=T
REPO_CUST_NEG185	Customize Commission Overview	/app/reporting/reportcomposer.nl?cr=-185&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG186	Customize Partner Commission Overview	/app/reporting/reportcomposer.nl?cr=186&e=T
REPO_CUST_NEG187	Customize Receivables by Customer	/app/reporting/reportcomposer.nl?cr=187&e=T
REPO_CUST_NEG188	Customize Billing and Revenue Summary	/app/reporting/reportcomposer.nl?cr=188&e=T
REPO_CUST_NEG195	Customize Budget Income Detail	/app/reporting/financialreportcomposer.nl?e=T&cr=195
REPO_CUST_NEG196	Customize Budget Income Statement	/app/reporting/financialreportcomposer.nl?e=T&cr=196
REPO_CUST_NEG197	Customize Budget vs. Actual	/app/reporting/financialreportcomposer.nl?e=T&cr=197
REPO_CUST_NEG198	Customize Comparative Balance Sheet	/app/reporting/financialreportcomposer.nl?e=T&cr=198
REPO_CUST_NEG199	Customize Comparative Income Statement	/app/reporting/financialreportcomposer.nl?e=T&cr=199
REPO_CUST_NEG200	Customize Income Statement	/app/reporting/financialreportcomposer.nl?e=T&cr=200
REPO_CUST_NEG201	Customize Cash Statement	/app/reporting/financialreportcomposer.nl?e=T&cr=201
REPO_CUST_NEG202	Customize Balance Sheet	/app/reporting/financialreportcomposer.nl?e=T&cr=202
REPO_CUST_NEG203	Customize Cash Flow Statement	/app/reporting/financialreportcomposer.nl?e=T&cr=203
REPO_CUST_NEG204	Customize Income Statement Detail	/app/reporting/financialreportcomposer.nl?e=T&cr=204
REPO_CUST_NEG205	Customize Cash Detail	/app/reporting/financialreportcomposer.nl?e=T&cr=205
REPO_CUST_NEG206	Customize Balance Sheet Detail	/app/reporting/financialreportcomposer.nl?e=T&cr=206
REPO_CUST_NEG209	Customize Open Return Authorizations	/app/reporting/reportcomposer.nl?cr=209&e=T
REPO_CUST_NEG210	Customize Return Authorizations Pending Receipt	/app/reporting/reportcomposer.nl?cr=210&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG211	Customize Return Authorizations Register	/app/reporting/reportcomposer.nl?cr=-211&e=T
REPO_CUST_NEG212	Customize Time Entry Exceptions	/app/reporting/reportcomposer.nl?cr=-212&e=T
REPO_CUST_NEG213	Customize Sales Orders by Historical Team Summary (Transaction Date)	/app/reporting/reportcomposer.nl?cr=-213&e=T
REPO_CUST_NEG214	Customize Sales Orders by Historical Team Detail (Transaction Date)	/app/reporting/reportcomposer.nl?cr=-214&e=T
REPO_CUST_NEG215	Customize Sales by Historical Team Summary (Transaction Date)	/app/reporting/reportcomposer.nl?cr=-215&e=T
REPO_CUST_NEG216	Customize Sales by Historical Team Detail (Transaction Date)	/app/reporting/reportcomposer.nl?cr=-216&e=T
REPO_CUST_NEG218	Customize Deferred Revenue Waterfall	/app/reporting/reportcomposer.nl?cr=-218&e=T
REPO_CUST_NEG221	Customize Transfer Order Register	/app/reporting/reportcomposer.nl?cr=-221&e=T
REPO_CUST_NEG222	Customize Purchase Order History	/app/reporting/reportcomposer.nl?cr=-222&e=T
REPO_CUST_NEG223	Customize Item Demand Forecast vs Sales	/app/reporting/reportcomposer.nl?cr=-223&e=T
REPO_CUST_NEG224	Customize Item Demand Plan by Item	/app/reporting/reportcomposer.nl?cr=-224&e=T
REPO_CUST_NEG225	Customize Demand History by Item	/app/reporting/reportcomposer.nl?cr=-225&e=T
REPO_CUST_NEG226	Customize Production Variances by Item	/app/reporting/reportcomposer.nl?cr=-226&e=T
REPO_CUST_NEG227	Customize Purchase Price Variances by Item	/app/reporting/reportcomposer.nl?cr=-227&e=T
REPO_CUST_NEG228	Customize Planned Standard Costs by Cost Version	/app/reporting/reportcomposer.nl?cr=-228&e=T
REPO_CUST_NEG231	Customize Intercompany Elimination	/app/reporting/reportcomposer.nl?cr=-231&e=T
REPO_CUST_NEG232	Customize Intercompany Reconciliation	/app/reporting/reportcomposer.nl?cr=-232&e=T
REPO_CUST_NEG233	Customize Open Estimates By Lines	/app/reporting/reportcomposer.nl?cr=-233&e=T
REPO_CUST_NEG234	Customize Open Estimates By Lines Detail	/app/reporting/reportcomposer.nl?cr=-234&e=T
REPO_CUST_NEG235	Customize Revenue Reclassification Detail	/app/reporting/reportcomposer.nl?e=T&cr=-235

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG236	Customize Revenue Reclassification	/app/reporting/reportcomposer.nl?e=T&cr=-236
REPO_CUST_NEG239	Customize Allocated vs. Actual Hours by Resource	/app/reporting/reportcomposer.nl?cr=-239&e=T
REPO_CUST_NEG240	Customize Allocated Utilization by Resource	/app/reporting/reportcomposer.nl?cr=-240&e=T
REPO_CUST_NEG241	Customize Commission Overview Detail	/app/reporting/reportcomposer.nl?cr=-241&e=T
REPO_CUST_NEG242	Customize Partner Commission Overview Detail	/app/reporting/reportcomposer.nl?cr=-242&e=T
REPO_CUST_NEG245	Customize GL Audit Numbering	/app/reporting/reportcomposer.nl?cr=-245&e=T
REPO_CUST_NEG246	Customize Planned Utilization by Resource	/app/reporting/reportcomposer.nl?cr=-246&e=T
REPO_CUST_NEG247	Customize Actual Utilization by Resource	/app/reporting/reportcomposer.nl?cr=-247&e=T
REPO_CUST_NEG248	Customize Allocated Utilization by Project	/app/reporting/reportcomposer.nl?cr=-248&e=T
REPO_CUST_NEG249	Customize Planned Utilization by Project	/app/reporting/reportcomposer.nl?cr=-249&e=T
REPO_CUST_NEG250	Customize Actual Utilization by Project	/app/reporting/reportcomposer.nl?cr=-250&e=T
REPO_CUST_NEG253	Customize Project Billing Report	/app/reporting/reportcomposer.nl?e=T&cr=-253
REPO_CUST_NEG254	Customize Project Cost Budget vs. Actual	/app/reporting/reportcomposer.nl?cr=-254&e=T
REPO_CUST_NEG255	Customize Project Billing Budget vs. Actual	/app/reporting/reportcomposer.nl?cr=-255&e=T
REPO_CUST_NEG256	Customize Project Task Cost Budget vs. Actual	/app/reporting/reportcomposer.nl?cr=-256&e=T
REPO_CUST_NEG257	Customize Project Task Billing Budget vs. Actual	/app/reporting/reportcomposer.nl?cr=-257&e=T
REPO_CUST_NEG258	Customize Project Profitability by Month	/app/reporting/reportcomposer.nl?cr=-258&e=T
REPO_CUST_NEG259	Customize Deferred Revenue Reclassification Activity	/app/reporting/reportcomposer.nl?e=T&cr=-259
REPO_CUST_NEG260	Customize Deferred Revenue Reclassification	/app/reporting/reportcomposer.nl?e=T&cr=-260
REPO_CUST_NEG265	Customize Revenue Contract Activity	/app/reporting/reportcomposer.nl?e=T&cr=-265
REPO_CUST_NEG273	Customize Project Cost Budget vs. Actual Detail	/app/reporting/reportcomposer.nl?cr=-273&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG274	Customize Project Billing Budget vs. Actual Detail	/app/reporting/reportcomposer.nl?cr=274&e=T
REPO_CUST_NEG275	Customize Project Profitability Detail	/app/reporting/reportcomposer.nl?cr=275&e=T
REPO_CUST_NEG277	Customize Deferred Revenue Rollforward	/app/reporting/reportcomposer.nl?e=T&cr=277
REPO_CUST_NEG278	Customize Deferred Revenue Rollforward Detail	/app/reporting/reportcomposer.nl?e=T&cr=278
REPO_CUST_NEG279	Customize Deferred Revenue Rollforward Customer	/app/reporting/reportcomposer.nl?e=T&cr=279
REPO_CUST_NEG280	Customize Campaign Clickthrough Detail	/app/reporting/reportcomposer.nl?cr=280&e=T
REPO_CUST_NEG281	Customize Campaign Clickthrough Summary	/app/reporting/reportcomposer.nl?cr=281&e=T
REPO_CUST_NEG282	Customize CTA Balance Audit	/app/reporting/financialreportcomposer.nl?e=T&cr=282
REPO_CUST_NEG284	Customize Project Profitability	/app/reporting/reportcomposer.nl?cr=284&e=T
REPO_CUST_NEG285	Customize Project Charges Forecast	/app/reporting/reportcomposer.nl?cr=285&e=T
REPO_CUST_NEG289	Customize Deferred Revenue Reclassification	/app/reporting/reportcomposer.nl?e=T&cr=289
REPO_CUST_NEG290	Customize Deferred Revenue Detail	/app/reporting/reportcomposer.nl?cr=290&e=T
REPO_CUST_NEG291	Customize Deferred Revenue by Customer Summary	/app/reporting/reportcomposer.nl?cr=291&e=T
REPO_CUST_NEG292	Customize Deferred Revenue Rollforward	/app/reporting/reportcomposer.nl?e=T&cr=292
REPO_CUST_NEG293	Customize Deferred Revenue Rollforward Detail	/app/reporting/reportcomposer.nl?e=T&cr=293
REPO_CUST_NEG294	Customize Deferred Revenue Rollforward Customer	/app/reporting/reportcomposer.nl?e=T&cr=294
REPO_CUST_NEG295	Customize Deferred Revenue Waterfall Summary	/app/reporting/reportcomposer.nl?cr=295&e=T
REPO_CUST_NEG296	Customize Monthly Recurring Revenue	/app/reporting/reportcomposer.nl?cr=296&e=T
REPO_CUST_NEG297	Customize Billings To Date	/app/reporting/reportcomposer.nl?cr=297&e=T
REPO_CUST_NEG298	Customize Stock Ledger	/app/reporting/reportcomposer.nl?cr=298&e=T
REPO_CUST_NEG299	Customize Total Contract Value	/app/reporting/reportcomposer.nl?cr=299&e=T

Task ID	Page Label in NetSuite	URL
REPO_CUST_NEG306	Customize Time-Off Balance Details	/app/reporting/reportcomposer.nl?cr=-306&e=T
REPO_CUST_NEG307	Customize Time-Off Balance Summary	/app/reporting/reportcomposer.nl?cr=-307&e=T
REPO_CUST_NEG308	Customize Available Time-Off	/app/reporting/reportcomposer.nl?cr=-308&e=T
REPO_CUST_PAGE	All Saved Reports	/app/reporting/savedreports.nl
REPO_EMAIL	Email Report	/app/accounting/reports/emailContact.nl
REPO_FINANCIALLAYOUTS	Financial Row Layouts	/app/reporting/financiallayouts.nl
REPO_GST34	GST34 Worksheet	/app/accounting/reports/intl/ca/gst34.nl
REPO_JPTAXFORM	Tax Form	/app/accounting/reports/intl/jptaxform.nl
REPO_JP_BS	Japanese Balance Sheet	/app/accounting/reports/intl/jp/jpbalanceSheet.nl
REPO_JP_CF	Japanese Cash Flow Statement	/app/accounting/reports/intl/jp/jpcashflow.nl
REPO_JP_EQUITY_CHANGE	Japanese Equity Change Report	/app/accounting/reports/intl/jp/jpequitychange.nl
REPO_JP_FR_PACKAGE	Japanese Financial Report Package	/app/accounting/reports/intl/jp/jpfrpackage.nl
REPO_JP_IS	Japanese Income Statement	/app/accounting/reports/intl/jp/jpincomestatement.nl
REPO_NEG100	Case Activity by Support Rep	/app/reporting/reportrunner.nl?cr=-100
REPO_NEG101	Case Activity by Support Rep Detail	/app/reporting/reportrunner.nl?cr=-101
REPO_NEG105	Utilization by Employee Summary (deprecated)	/app/reporting/reportrunner.nl?cr=-105
REPO_NEG106	Current Backlog by Resource	/app/reporting/reportrunner.nl?cr=-106
REPO_NEG108	Sales Orders by Sales Team	/app/reporting/reportrunner.nl?cr=-108
REPO_NEG109	Sales Orders by Sales Team Detail	/app/reporting/reportrunner.nl?cr=-109
REPO_NEG110	Sales by Sales Team	/app/reporting/reportrunner.nl?cr=-110
REPO_NEG111	Sales by Sales Team Detail	/app/reporting/reportrunner.nl?cr=-111
REPO_NEG112	Aggregated Sales Orders by Customer Summary	/app/reporting/reportrunner.nl?cr=-112
REPO_NEG113	Aggregated Sales Orders by Item Summary	/app/reporting/reportrunner.nl?cr=-113
REPO_NEG114	Internal Search	/app/reporting/reportrunner.nl?cr=-114
REPO_NEG115	Internal Search Detail	/app/reporting/reportrunner.nl?cr=-115
REPO_NEG116	Customer Aging History	/app/reporting/reportrunner.nl?cr=-116



Task ID	Page Label in NetSuite	URL
REPO_NEG118	Estimated Profitability by Job	/app/reporting/reportrunner.nl?cr=-118
REPO_NEG119	Reconciliation Detail	/app/reporting/reportrunner.nl?cr=-119
REPO_NEG120	Sales Tax on Sales	/app/reporting/reportrunner.nl?cr=-120
REPO_NEG121	Sales Tax on Sales Detail	/app/reporting/reportrunner.nl?cr=-121
REPO_NEG122	Campaign ROI Analysis	/app/reporting/reportrunner.nl?cr=-122
REPO_NEG123	Sales by Lead Source Detail	/app/reporting/reportrunner.nl?cr=-123
REPO_NEG124	Sales by Lead Source	/app/reporting/reportrunner.nl?cr=-124
REPO_NEG125	PST on Sales Detail	/app/reporting/reportrunner.nl?cr=-125
REPO_NEG126	PST on Purchases Detail	/app/reporting/reportrunner.nl?cr=-126
REPO_NEG127	PST on Sales	/app/reporting/reportrunner.nl?cr=-127
REPO_NEG128	PST on Purchases	/app/reporting/reportrunner.nl?cr=-128
REPO_NEG129	Reconciliation	/app/reporting/reportrunner.nl?cr=-129
REPO_NEG130	Bank Register	/app/reporting/reportrunner.nl?cr=-130
REPO_NEG131	Sales Tax Analysis	/app/reporting/reportrunner.nl?cr=-131
REPO_NEG132	Realized Exchange Rate Gains and Losses	/app/reporting/reportrunner.nl?cr=-132
REPO_NEG133	Unrealized Exchange Rate Gains and Losses	/app/reporting/reportrunner.nl?cr=-133
REPO_NEG134	Sales by Paid Keyword Detail	/app/reporting/reportrunner.nl?cr=-134
REPO_NEG135	Sales by Paid Keyword	/app/reporting/reportrunner.nl?cr=-135
REPO_NEG136	Leads by Paid Keyword Detail	/app/reporting/reportrunner.nl?cr=-136
REPO_NEG137	Leads by Paid Keyword	/app/reporting/reportrunner.nl?cr=-137
REPO_NEG138	Closed Issues	/app/reporting/reportrunner.nl?cr=-138
REPO_NEG139	Closed Issues Detail	/app/reporting/reportrunner.nl?cr=-139
REPO_NEG140	Open Issues	/app/reporting/reportrunner.nl?cr=-140
REPO_NEG141	Open Issues Detail	/app/reporting/reportrunner.nl?cr=-141
REPO_NEG144	Paid Employee Commission	/app/reporting/reportrunner.nl?cr=-144
REPO_NEG145	Paid Employee Commission Detail	/app/reporting/reportrunner.nl?cr=-145
REPO_NEG146	Paid Partner Commission	/app/reporting/reportrunner.nl?cr=-146
REPO_NEG147	Paid Partner Commission Detail	/app/reporting/reportrunner.nl?cr=-147
REPO_NEG148	Sales Order Revenue Forecast	/app/reporting/reportrunner.nl?cr=-148
REPO_NEG149	Sales Order Revenue Forecast Detail	/app/reporting/reportrunner.nl?cr=-149
REPO_NEG150	Forecast vs. Quota by Class	/app/reporting/reportrunner.nl?cr=-150
REPO_NEG151	Forecast Accuracy	/app/reporting/reportrunner.nl?cr=-151

Task ID	Page Label in NetSuite	URL
REPO_NEG152	Forecast Accuracy (ALT_SALES)	/app/reporting/reportrunner.nl?cr=-152
REPO_NEG153	Calculated Forecast Accuracy	/app/reporting/reportrunner.nl?cr=-153
REPO_NEG154	Calculated Forecast Accuracy (ALT_SALES)	/app/reporting/reportrunner.nl?cr=-154
REPO_NEG155	Forecast vs. Quota by Department	/app/reporting/reportrunner.nl?cr=-155
REPO_NEG156	Forecast vs. Quota by Location	/app/reporting/reportrunner.nl?cr=-156
REPO_NEG157	ALT_SALES Forecast vs. Quota by Class	/app/reporting/reportrunner.nl?cr=-157
REPO_NEG158	ALT_SALES Forecast vs. Quota by Department	/app/reporting/reportrunner.nl?cr=-158
REPO_NEG159	ALT_SALES Forecast vs. Quota by Location	/app/reporting/reportrunner.nl?cr=-159
REPO_NEG160	Forecast Accuracy	/app/reporting/reportrunner.nl?cr=-160
REPO_NEG161	Calculated Forecast Accuracy	/app/reporting/reportrunner.nl?cr=-161
REPO_NEG162	Earned Value by Job	/app/reporting/reportrunner.nl?cr=-162
REPO_NEG175	Service Fees	/app/reporting/reportrunner.nl?cr=-175
REPO_NEG178	Utilization by Employee (deprecated)	/app/reporting/reportrunner.nl?cr=-178
REPO_NEG179	Deferred Revenue By Customer	/app/reporting/reportrunner.nl?cr=-179
REPO_NEG180	Deferred Revenue By Item	/app/reporting/reportrunner.nl?cr=-180
REPO_NEG181	Deferred Revenue By State	/app/reporting/reportrunner.nl?cr=-181
REPO_NEG182	Revenue By Customer	/app/reporting/reportrunner.nl?cr=-182
REPO_NEG183	Revenue By Item	/app/reporting/reportrunner.nl?cr=-183
REPO_NEG184	Revenue By State	/app/reporting/reportrunner.nl?cr=-184
REPO_NEG185	Commission Overview	/app/reporting/reportrunner.nl?cr=-185
REPO_NEG186	Partner Commission Overview	/app/reporting/reportrunner.nl?cr=-186
REPO_NEG187	Receivables by Customer	/app/reporting/reportrunner.nl?cr=-187
REPO_NEG188	Billing and Revenue Summary	/app/reporting/reportrunner.nl?cr=-188
REPO_NEG195	Budget Income Statement Detail	/app/reporting/reportrunner.nl?cr=-195
REPO_NEG196	Budget Income Statement	/app/reporting/reportrunner.nl?cr=-196
REPO_NEG197	Budget vs. Actual	/app/reporting/reportrunner.nl?cr=-197
REPO_NEG198	Comparative Balance Sheet	/app/reporting/reportrunner.nl?cr=-198
REPO_NEG199	Comparative Income Statement	/app/reporting/reportrunner.nl?cr=-199
REPO_NEG200	Income Statement	/app/reporting/reportrunner.nl?cr=-200

Task ID	Page Label in NetSuite	URL
REPO_NEG201	Cash Statement	/app/reporting/reportrunner.nl?cr=-201
REPO_NEG202	Balance Sheet	/app/reporting/reportrunner.nl?cr=-202
REPO_NEG203	Cash Flow Statement	/app/reporting/reportrunner.nl?cr=-203
REPO_NEG204	Income Statement Detail	/app/reporting/reportrunner.nl?cr=-204
REPO_NEG205	Cash Statement Detail	/app/reporting/reportrunner.nl?cr=-205
REPO_NEG206	Balance Sheet Detail	/app/reporting/reportrunner.nl?cr=-206
REPO_NEG209	Open Return Authorizations	/app/reporting/reportrunner.nl?cr=-209
REPO_NEG210	Return Authorizations Pending Receipt	/app/reporting/reportrunner.nl?cr=-210
REPO_NEG211	Return Authorizations Register	/app/reporting/reportrunner.nl?cr=-211
REPO_NEG212	Time Entry Exceptions	/app/reporting/reportrunner.nl?cr=-212
REPO_NEG213	Sales Orders by Historical Team (Transaction Date)	/app/reporting/reportrunner.nl?cr=-213
REPO_NEG214	Sales Orders by Item Detail (Transaction Date)	/app/reporting/reportrunner.nl?cr=-214
REPO_NEG215	Sales by Historical Team (Transaction Date)	/app/reporting/reportrunner.nl?cr=-215
REPO_NEG216	Sales by Item Detail (Transaction Date)	/app/reporting/reportrunner.nl?cr=-216
REPO_NEG218	Deferred Revenue Waterfall Detail	/app/reporting/reportrunner.nl?cr=-218
REPO_NEG221	Transfer Order Register	/app/reporting/reportrunner.nl?cr=-221
REPO_NEG222	Purchase Order History	/app/reporting/reportrunner.nl?cr=-222
REPO_NEG223	Item Demand Forecast vs Actual	/app/reporting/reportrunner.nl?cr=-223
REPO_NEG224	Item Demand Plan by Item	/app/reporting/reportrunner.nl?cr=-224
REPO_NEG225	Demand History by Item	/app/reporting/reportrunner.nl?cr=-225
REPO_NEG226	Production Variances by Item	/app/reporting/reportrunner.nl?cr=-226
REPO_NEG227	Purchase Price Variances by Item	/app/reporting/reportrunner.nl?cr=-227
REPO_NEG228	Planned Standard Costs by Cost Version	/app/reporting/reportrunner.nl?cr=-228
REPO_NEG231	Intercompany Elimination	/app/reporting/reportrunner.nl?cr=-231
REPO_NEG232	Intercompany Reconciliation	/app/reporting/reportrunner.nl?cr=-232
REPO_NEG233	Open Estimates By Lines	/app/reporting/reportrunner.nl?cr=-233
REPO_NEG234	Open Estimates By Lines Detail	/app/reporting/reportrunner.nl?cr=-234
REPO_NEG235	Revenue Reclassification Detail	/app/reporting/reportrunner.nl?cr=-235
REPO_NEG236	Revenue Reclassification	/app/reporting/reportrunner.nl?cr=-236



Task ID	Page Label in NetSuite	URL
REPO_NEG237	Employee Change History	/app/reporting/reportrunner.nl?cr=-237
REPO_NEG239	Allocated vs. Actual Hours by Resource	/app/reporting/reportrunner.nl?cr=-239
REPO_NEG240	Allocated Utilization by Resource	/app/reporting/reportrunner.nl?cr=-240
REPO_NEG241	Commission Overview Detail	/app/reporting/reportrunner.nl?cr=-241
REPO_NEG242	Partner Commission Overview Detail	/app/reporting/reportrunner.nl?cr=-242
REPO_NEG245	GL Audit Numbering	/app/reporting/reportrunner.nl?cr=-245
REPO_NEG246	Planned Utilization by Resource	/app/reporting/reportrunner.nl?cr=-246
REPO_NEG247	Actual Utilization by Resource	/app/reporting/reportrunner.nl?cr=-247
REPO_NEG248	Allocated Utilization by Project	/app/reporting/reportrunner.nl?cr=-248
REPO_NEG249	Planned Utilization by Project	/app/reporting/reportrunner.nl?cr=-249
REPO_NEG250	Actual Utilization by Project	/app/reporting/reportrunner.nl?cr=-250
REPO_NEG253	Project Billing Report	/app/reporting/reportrunner.nl?cr=-253
REPO_NEG254	Project Cost Budget vs. Actual	/app/reporting/reportrunner.nl?cr=-254
REPO_NEG255	Project Billing Budget vs. Actual	/app/reporting/reportrunner.nl?cr=-255
REPO_NEG256	Project Task Cost Budget vs. Actual	/app/reporting/reportrunner.nl?cr=-256
REPO_NEG257	Project Task Billing Budget vs. Actual	/app/reporting/reportrunner.nl?cr=-257
REPO_NEG258	Project Profitability by Month	/app/reporting/reportrunner.nl?cr=-258
REPO_NEG259	Deferred Revenue Reclassification Activity	/app/reporting/reportrunner.nl?cr=-259
REPO_NEG260	Deferred Revenue Reclassification	/app/reporting/reportrunner.nl?cr=-260
REPO_NEG265	Revenue Contract Activity	/app/reporting/reportrunner.nl?cr=-265
REPO_NEG271	Employee Payroll Item History	/app/reporting/reportrunner.nl?cr=-271
REPO_NEG273	Project Cost Budget vs. Actual Detail	/app/reporting/reportrunner.nl?cr=-273
REPO_NEG274	Project Billing Budget vs. Actual Detail	/app/reporting/reportrunner.nl?cr=-274
REPO_NEG275	Project Profitability Detail	/app/reporting/reportrunner.nl?cr=-275
REPO_NEG277	Deferred Revenue Rollforward	/app/reporting/reportrunner.nl?cr=-277
REPO_NEG278	Deferred Revenue Rollforward Detail	/app/reporting/reportrunner.nl?cr=-278
REPO_NEG279	Deferred Revenue Rollforward Customer	/app/reporting/reportrunner.nl?cr=-279
REPO_NEG280	Campaign Clickthrough Detail	/app/reporting/reportrunner.nl?cr=-280

Task ID	Page Label in NetSuite	URL
REPO_NEG281	Campaign Clickthrough Summary	/app/reporting/reportrunner.nl?cr=-281
REPO_NEG282	CTA Balance Audit	/app/reporting/reportrunner.nl?cr=-282
REPO_NEG284	Project Profitability	/app/reporting/reportrunner.nl?cr=-284
REPO_NEG285	Project Charges Forecast	/app/reporting/reportrunner.nl?cr=-285
REPO_NEG289	Deferred Revenue Reclassification	/app/reporting/reportrunner.nl?cr=-289
REPO_NEG290	Revenue Recognition Forecast Detail	/app/reporting/reportrunner.nl?cr=-290
REPO_NEG291	Revenue Recognition Forecast	/app/reporting/reportrunner.nl?cr=-291
REPO_NEG292	Deferred Revenue Rollforward	/app/reporting/reportrunner.nl?cr=-292
REPO_NEG293	Deferred Revenue Rollforward Detail	/app/reporting/reportrunner.nl?cr=-293
REPO_NEG294	Deferred Revenue Rollforward Customer	/app/reporting/reportrunner.nl?cr=-294
REPO_NEG295	Deferred Revenue Waterfall	/app/reporting/reportrunner.nl?cr=-295
REPO_NEG296	Monthly Recurring Revenue	/app/reporting/reportrunner.nl?cr=-296
REPO_NEG297	Billings To Date	/app/reporting/reportrunner.nl?cr=-297
REPO_NEG298	Stock Ledger	/app/reporting/reportrunner.nl?cr=-298
REPO_NEG299	Total Contract Value	/app/reporting/reportrunner.nl?cr=-299
REPO_NEG306	Time-Off Balance Details	/app/reporting/reportrunner.nl?cr=-306
REPO_NEG307	Time-Off Balance Summary	/app/reporting/reportrunner.nl?cr=-307
REPO_NEG308	Available Time-Off	/app/reporting/reportrunner.nl?cr=-308
REPO_NEWFINANCIALREPORT	New Financial Report	/app/reporting/newfinancialreport.nl
REPO_QUICKREPORT	New Report	/app/reporting/quickreports.nl
REPO_RECENT	Recent Reports	/app/common/otherlists/recentrecords.nl
REPO_REGISTER_ACCTPAY	Accounts Payable Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=AcctPay
REPO_REGISTER_ACCTREC	Accounts Receivable Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=AcctRec
REPO_REGISTER_BANK	Bank Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=Bank
REPO_REGISTER_COGS	COGS Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=COGS
REPO_REGISTER_CREDCARD	Credit Card Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=CredCard

Task ID	Page Label in NetSuite	URL
REPO_REGISTER_DEFEREXPENSE	Deferred Expense Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=DeferExpense
REPO_REGISTER_DEFERREVENUE	Deferred Revenue Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=DeferRevenue
REPO_REGISTER_EQUITY	Equity Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=Equity
REPO_REGISTER_EXPENSE	Expense Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=Expense
REPO_REGISTER_FIXEDASSET	Fixed Asset Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=FixedAsset
REPO_REGISTER_INCOME	Income Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=Income
REPO_REGISTER_LONGTERMLIAB	Long Team Liability Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=LongTermLiab
REPO_REGISTER_NONPOSTING	Non-Posting Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=NonPosting
REPO_REGISTER_OTHASSET	Other Asset Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=OthAsset
REPO_REGISTER_OTHCURRASSET	Other Current Asset Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=OthCurrAsset
REPO_REGISTER_OTHCURRLIAB	Other Current Liability Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=OthCurrLiab
REPO_REGISTER_OTHEXPENSE	Register - Other Expense	/app/reporting/reportrunner.nl?acctid=&showStartBalances=F
REPO_REGISTER_OTHINCOME	Other Income Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=OthIncome
REPO_REGISTER_UNBILLEDREC	Unbilled Receivable Register	/app/reporting/reportrunner.nl?reporttype=REGISTER&accttype=UnbilledRec
SRCH_ACCOUNT	Search Account	/app/common/search/search.nl?searchtype=Account
SRCH_ACCOUNTINGBOOK	Search Accounting Book	/app/common/search/search.nl?searchtype=AccountingBook
SRCH_ACCOUNTINGPERIOD	Search Accounting Period	/app/common/search/search.nl?searchtype=AccountingPeriod



Task ID	Page Label in NetSuite	URL
SRCH_ACCOUNTINGTRANSACTION	Search Accounting Transactions	/app/common/search/search.nl?searchtype=AccountingTransaction
SRCH_ACTIVITY	Search Activities	/app/common/search/search.nl?searchtype=Activity
SRCH_AMORTIZATIONSCHEDULE	Search Amortization Schedules	/app/common/search/search.nl?searchtype=AmortizationSchedule
SRCH_APPDEF	Search App Definitions	/app/common/search/search.nl?searchtype=AppDefinition
SRCH_APPPKG	Search App Packages	/app/common/search/search.nl?searchtype=AppPackage
SRCH_AUDITTRAIL	Search Audit Trail	/app/common/search/search.nl?searchtype=AuditTrail
SRCH_BILLINGACCOUNT	Search Billing Account	/app/common/search/search.nl?searchtype=BillingAccount
SRCH_BILLINGCLASS	Search Billing Class	/app/common/search/search.nl?searchtype=BillingClass
SRCH_BILLINGRULE	Search Billing Rule	/app/common/search/search.nl?searchtype=BillingRule
SRCH_BILLINGSCHEDULE	Search Billing Schedule	/app/common/search/search.nl?searchtype=BillingSchedule
SRCH_BILLING_GROUPS	Billing Groups	/app/common/search/search.nl?searchtype=CRMGroup&adv=T&CRMGroup_GROUPTYPE=CustJob&CRMGroup_ISBILLINGGROUP=T
SRCH_BILLOFDISTRIBUTION	Search Bill Of Distribution	/app/common/search/search.nl?searchtype=BillOfDistribution
SRCH_BILLRUN	Search Billing Operations	/app/common/search/search.nl?searchtype=BillRun
SRCH_BILLRUNSCHEDULE	Search Billing Operation Schedules	/app/common/search/search.nl?searchtype=BillRunSchedule
SRCH_BIN	Search Bin	/app/common/search/search.nl?searchtype=BinNumber
SRCH_BINNUMBER	Search Bin	/app/common/search/search.nl?searchtype=BinNumber
SRCH_BUDGET	Search Set Up Budget	/app/common/search/search.nl?searchtype=Budget
SRCH_BUDGETRATES	Search Budget Rates	/app/common/search/search.nl?searchtype=BudgetExchangeRate
SRCH_CALENDAR	Search Calendar/Event	/app/common/search/search.nl?searchtype=Calendar
SRCH_CALL	Search Calls	/app/common/search/search.nl?searchtype=Call

Task ID	Page Label in NetSuite	URL
SRCH_CAMPAIGN	Search Campaigns	/app/common/search/search.nl?searchtype=Campaign
SRCH_CAMPAIGNTEMPLATE	Search All Campaign Templates	/app/common/search/search.nl?searchtype=CampaignTemplate
SRCH_CASE	Search Case	/app/common/search/search.nl?searchtype=Case
SRCH_CHARGE	Search Charge	/app/common/search/search.nl?searchtype=Charge
SRCH_CHARGERULE	Search Charge Rule	/app/common/search/search.nl?searchtype=ChargeRule
SRCH_CLASS	Search Class	/app/common/search/search.nl?searchtype=Class
SRCH_CLASSSEGMENTAPPING	Search Class Mapping	/app/common/search/search.nl?searchtype=ClassSegmentMapping
SRCH_COMMERCECATALOG	Search Commerce Category Catalogs	/app/common/search/search.nl?searchtype=commercecatalog
SRCH_COMMERCECATEGORY	Search Commerce Categories	/app/common/search/search.nl?searchtype=commercecategory
SRCH_COMMISSIONABLEITEM	Search Commissionable Items	/app/common/search/search.nl?searchtype=Transaction
SRCH_COMMISSIONOVERVIEW	Search Commission Overview	/app/common/search/search.nl?searchtype=Transaction
SRCH_COMPANY	Search All Companies	/app/common/search/search.nl?searchtype=Company
SRCH_COMPETITOR	Search Competitors	/app/common/search/search.nl?searchtype=Competitor
SRCH_CONSOLEXCHANGERATE	Consolidated Exchange Rates	/app/common/search/search.nl?searchtype=ConsolExchangeRate
SRCH_CONSOLRATES	Search Consolidated Rates	/app/common/search/search.nl?searchtype=ConsolExchangeRate
SRCH_CONTACT	Search Contact	/app/common/search/search.nl?searchtype=Contact
SRCH_COSTCATEGORY	Cost Category	/app/common/search/search.nl?searchtype=CostCategory
SRCH_COUPONCODE	Search Coupon Code	/app/common/search/search.nl?searchtype=CouponCode
SRCH_CRMGROUP	Search Group	/app/common/search/search.nl?searchtype=CRMGroup
SRCH_CUSTBAL	Customers by Open A/R Balance	/app/common/search/searchresults.nl?searchid=-72
SRCH_CUSTOMER	Search Customer	/app/common/search/search.nl?searchtype=Customer



Task ID	Page Label in NetSuite	URL
SRCH_CUSTOMSEGMENT	Search Custom Segments	/app/common/search/search.nl?searchtype=CustomSegment
SRCH_CUSTOVDBAL	Customers by Outstanding Balance	/app/common/search/searchresults.nl?searchid=-73
SRCH_DELETEDRECORD	Search Deleted Records	/app/common/search/search.nl?searchtype=DeletedRecord
SRCH_DEPARTMENT	Search Department	/app/common/search/search.nl?searchtype=Department
SRCH_DEPTSEGMENTMAPPING	Search Department Mapping	/app/common/search/search.nl?searchtype=DeptSegmentMapping
SRCH_DISTRIBUTIONNETWORK	Search Distribution Network	/app/common/search/search.nl?searchtype=DistributionNetwork
SRCH_DOCUMENT	Search File Cabinet	/app/common/search/search.nl?searchtype=Document
SRCH_DRIVERSLICENSE	Search Driver's Licenses	/app/common/search/search.nl?searchtype=DriversLicense
SRCH_EMAILTEMPLATE	Search All Email Templates	/app/common/search/search.nl?searchtype=EmailTemplate
SRCH_EMPLOYEE	Search Employee	/app/common/search/search.nl?searchtype=Employee
SRCH_EMPLOYEEPAYROLLITEM	Search Employee Payroll Items	/app/common/search/search.nl?searchtype=EmployeePayrollItem&rectype
SRCH_ENTITY	Search All Entities	/app/common/search/search.nl?searchtype=Entity
SRCH_ENTITYACCOUNTMAPPING	Search Entity Account Mapping	/app/common/search/search.nl?searchtype=EntityAccountMapping
SRCH_FAIRVALUEFORMULA	Search Fair Value Formulas	/app/common/search/search.nl?searchtype=FairValueFormula
SRCH_FAIRVALUEPRICE	Search Fair Value Price Form	/app/common/search/search.nl?searchtype=FairValuePrice
SRCH_FAXTEMPLATE	Search All Fax Templates	/app/common/search/search.nl?searchtype=FaxTemplate
SRCH_FIRSTVISIT	Search First Site Visit	/app/common/search/search.nl?searchtype=FirstVisit
SRCH_FISCALCALENDAR	Search Fiscal Calendars	/app/common/search/search.nl?searchtype=FiscalCalendar
SRCH_FOLDER	Search Folder	/app/common/search/search.nl?searchtype=Folder
SRCH_GENERICRESOURCE	Search Generic Resources	/app/common/search/search.nl?searchtype=GenericResource
SRCH_GIFTCERTIFICATE	Search Gift Certificates	/app/common/search/search.nl?searchtype=GiftCertificate

Task ID	Page Label in NetSuite	URL
SRCH_GLLINESAUDITLOG	Search Custom GL Lines Plug-in Audit Log	/app/common/search/search.nl?searchtype=GLLinesAuditLog
SRCH_GLOBALACCOUNTMAPPING	Search Global Account Mapping	/app/common/search/search.nl?searchtype=GlobalAccountMapping
SRCH_GOVERNMENTISSUEDIDTYPE	Search Government-Issued ID Types	/app/common/search/search.nl?searchtype=GovernmentIssuedIdType
SRCH_HCMJOB	Search HCM Job	/app/common/search/search.nl?searchtype=HCMJob
SRCH_INFOITEM	Search Information Items	/app/common/search/search.nl?id=-2540&e=T
SRCH_INFOITEMFORM	Search Published Forms	/app/common/search/search.nl?id=-2541&e=T
SRCH_INVCOSTTEMPLATE	Search Inventory Cost Template	/app/common/search/search.nl?searchtype=InvCostTemplate
SRCH_INVENTORYNUMBER	Search Inventory Number	/app/common/search/search.nl?searchtype=InventoryNumber
SRCH_INVENTORYNUMBERBIN	Search Inventory Number Bin	/app/common/search/search.nl?searchtype=InventoryNumberBin
SRCH_IPRESTRICTIONS	Login Restrictions	/app/common/search/search.nl?searchtype=IPRestrictions
SRCH_ISSUE	Search Issue	/app/common/search/search.nl?searchtype=Issue
SRCH_ITEM	Search Item	/app/common/search/search.nl?searchtype=Item
SRCH_ITEMACCOUNTMAPPING	Search Item Account Mapping	/app/common/search/search.nl?searchtype=ItemAccountMapping
SRCH_ITEMDEMANDPLAN	Search Item Demand Plan	/app/common/search/search.nl?searchtype=ItemDemandPlan
SRCH_ITEMREVENUECATEGORY	Search Item Revenue Category Form	/app/common/search/search.nl?searchtype=ItemRevenueCategory
SRCH_ITEMSUPPLYPLAN	Search Item Supply Plan	/app/common/search/search.nl?searchtype=ItemSupplyPlan
SRCH_ITEM_REVISION	Search Item Revision	/app/common/search/search.nl?searchtype=ItemRevision
SRCH_JOB	Search Job	/app/common/search/search.nl?searchtype=Job
SRCH_JOBREQUISITION	Search Job Requisition	/app/common/search/search.nl?searchtype=JobRequisition
SRCH_KUDOS	Search Kudos	/app/common/search/search.nl?searchtype=Kudos
SRCH_LEAD	Search Leads	/app/common/search/search.nl?searchtype=Customer&Customer_STAGE=LEAD

Task ID	Page Label in NetSuite	URL
SRCH_LOCASSIGNCONF	Configurations Search	/app/common/search/search.nl?searchtype=LocAssignConf
SRCH_LOCATION	Search Location	/app/common/search/search.nl?searchtype=Location
SRCH_LOCATIONCOSTINGGROUP	Search Location Costing Group	/app/common/search/search.nl?searchtype=LocationCostingGroup
SRCH_LOCSEGMENTMAPPING	Search Location Mapping	/app/common/search/search.nl?searchtype=LocSegmentMapping
SRCH_LOGINAUDIT	Search Login Audit Trail	/app/common/search/search.nl?searchtype=LoginAudit
SRCH_MEDIAITEM	Search Media Items	/app/common/search/search.nl?searchtype=Document
SRCH_MEMDOC	Search Memorized Transactions	/app/common/search/search.nl?searchtype=MemDoc
SRCH_MESSAGE	Search Messages	/app/common/search/search.nl?searchtype=Message
SRCH_MFGCOSTTEMPLATE	Search Manufacturing Cost Template	/app/common/search/search.nl?searchtype=MfgCostTemplate
SRCH_MFGOPERATIONTASK	Search Manufacturing Operation Task	/app/common/search/search.nl?searchtype=MfgOperationTask
SRCH_MFGPLANNEDTIME	Search Manufacturing Planned Time	/app/common/search/search.nl?searchtype=MfgPlannedTime
SRCH_MFGROUTING	Search Manufacturing Routing	/app/common/search/search.nl?searchtype=MfgRouting
SRCH_NEXUS	Search Nexus	/app/common/search/search.nl?searchtype=Nexus
SRCH_NOTIFICATION	Search Notification	/app/common/search/search.nl?searchtype=Notification
SRCH_OAUTH_TOKENS	Search Access Tokens	/app/common/search/search.nl?searchtype=OAuthToken
SRCH_ONLINECASEFORM	Search Online Case Forms	/app/common/search/search.nl?searchtype=OnlineCaseForm
SRCH_ONLINELEADFORM	Search Online Lead Forms	/app/common/search/search.nl?searchtype=OnlineLeadForm
SRCH_ORGANIZATIONVALUE	Search Organization Values	/app/common/search/search.nl?searchtype=OrganizationValue
SRCH_OTHERGOVERNMENTISSUEDID	Search Other Government-Issued IDs	/app/common/search/search.nl?searchtype=OtherGovernmentIssuedId
SRCH_PARTNER	Search Partner	/app/common/search/search.nl?searchtype=Partner
SRCH_PASSPORT	Search Passports	/app/common/search/search.nl?searchtype=Passport

Task ID	Page Label in NetSuite	URL
SRCH_PAYMENTEVENT	Search Payment Events	/app/common/search/search.nl?searchtype=PaymentEvent
SRCH_PAYMETHODS	Search Payment Methods	/app/common/search/search.nl?searchtype=PaymentMethod
SRCH_PAYROLLITEM	Search Payroll Item	/app/common/search/search.nl?searchtype=PayrollItem
SRCH_PAYTERMS	Search Payment Terms	/app/common/search/search.nl?searchtype=Term
SRCH_PLANNEDSTANDARDCOST	Search Planned Standard Cost	/app/common/search/search.nl?searchtype=PlannedStandardCost
SRCH_POSITION	Search Position	/app/common/search/search.nl?searchtype=Position
SRCH_PRESCATEGORY	Search Categories	/app/common/search/search.nl?searchtype=SiteCategory
SRCH_PRICING	Search Pricing	/app/common/search/search.nl?searchtype=Pricing
SRCH_PROJECTBUDGET	Search Project Budget	/app/common/search/search.nl?searchtype=ProjectBudget
SRCH_PROJECTEXPENSETYPE	Search Project Expense Types	/app/common/search/search.nl?searchtype=ProjectExpenseType
SRCH_PROJECTRESOURCE	Search Project Resources	/app/common/search/search.nl?searchtype=ProjectResource
SRCH_PROJECTTASK	Search Project Tasks	/app/common/search/search.nl?searchtype=ProjectTask
SRCH_PROJECTTASKANDCRM_TASK	Search Project Tasks and CRM Tasks	/app/common/search/search.nl?searchtype=ProjectTaskAndCrmTask
SRCH_PROJECTTEMPLATE	Search Project Template	/app/common/search/search.nl?searchtype=ProjectTemplate
SRCH_PROMOTION	Search Promotion	/app/common/search/search.nl?searchtype=Promotion
SRCH_PROSPECT	Search Prospects	/app/common/search/search.nl?searchtype=Customer&Customer_STAGE=PROSPECT
SRCH_QUOTA	Search Establish Quota	/app/common/search/search.nl?searchtype=Quota
SRCH_RATEPLAN	Search Rate Plan	/app/common/search/search.nl?searchtype=RatePlan
SRCH_REGION	Search Region	/app/common/search/search.nl?searchtype=Region
SRCH_REPORT	Search Reports	/app/common/search/search.nl?searchtype=Report
SRCH_REVARRNGMESSAGE	Search Revenue Arrangement Message	/app/common/search/search.nl?searchtype=RevArrngMessage



Task ID	Page Label in NetSuite	URL
SRCH_REVENUEALLOCATIO NGROUP	Search Revenue Allocation Group	/app/common/search/search.nl? searchtype=RevAllocationGroup
SRCH_REVENUEELEMENT	Search Revenue Element	/app/common/search/search.nl? searchtype=RevenueElement
SRCH_REVENUEPLAN	Search Revenue Recognition Plan Form	/app/common/search/search.nl? searchtype=RevenuePlan
SRCH_REVENUERECOGNIT IONRULE	Search Revenue Recognition Rule Form	/app/common/search/search.nl? searchtype=RevRecRule
SRCH_REVREC	Search Revenue Recognition Schedules	/app/common/search/search.nl? searchtype=RevRecTemplate
SRCH_REVRECFIELDMAPPIN G	Search Revenue Field Mappings	/app/common/search/search.nl? searchtype=RevRecFieldMapping
SRCH_REVRECOGNITIONSCHE D	Search Revenue Recognition Schedules	/app/common/search/search.nl? searchtype=RevRecognitionSched
SRCH_ROLE	Search Role	/app/common/search/search.nl? searchtype=Role
SRCH_RSRCALLOCATION	Search Rsrc Allocations	/app/common/search/search.nl? searchtype=RsrcAllocation
SRCH_SALESCAMPAIGN	Search Sales Campaigns	/app/common/search/search.nl? searchtype=SalesCampaign
SRCH_SALESTERRITORIES	Sales Territory Search	/app/common/search/search.nl? searchtype=SalesTerritory
SRCH_SAVEDSEARCH	All Saved Searches	/app/common/search/savedsearchlist.nl
SRCH_SCHEDULEDSCRIPTI NSTANCE	Search Scheduled Script Instance	/app/common/search/search.nl? searchtype=ScheduledScriptInstance
SRCH_SCRIPT	Search Script	/app/common/search/search.nl? searchtype=Script
SRCH_SCRIPTDEPLOYMENT	Search Script Deployment	/app/common/search/search.nl? searchtype=ScriptDeployment
SRCH_SCRIPTNOTE	Search Script Execution Log	/app/common/search/search.nl? searchtype=ScriptNote
SRCH_SHIPITEM	Search Ship Item	/app/common/search/search.nl? searchtype=ShipItem
SRCH_SHIPMENTPACKAGE	Search Shipping Packages	/app/common/search/search.nl? searchtype=ShipmentPackage
SRCH_SHIPPARTPACKAGE	Search Shipping Partner Package	/app/common/search/search.nl? searchtype=ShipPartPackage
SRCH_SHIPPARTREGISTRA TION	Search Shipping Partner Registration	/app/common/search/search.nl? searchtype=ShipPartRegistration
SRCH_SHIPPARTSHIPMENT	Search Shipping Partner Shipment	/app/common/search/search.nl? searchtype=ShipPartShipment
SRCH_SHOPPINGCART	Search Shopping Cart	/app/common/search/search.nl? searchtype=ShoppingCart

Task ID	Page Label in NetSuite	URL
SRCH SOLUTION	Search Solution	/app/common/search/search.nl?searchtype=Solution
SRCH_STANDARDCOSTVERSION	Search Standard Cost Version	/app/common/search/search.nl?searchtype=StandardCostVersion
SRCH_SUBSCRIPTION	Search Subscriptions	/app/common/search/search.nl?searchtype=Subscription
SRCH_SUBSCRIPTIONCHAN GEORDER	Search Change Orders	/app/common/search/search.nl?searchtype=SubscriptionChangeOrder
SRCH_SUBSCRIPTIONLINE	Search Subscription Lines	/app/common/search/search.nl?searchtype=SubscriptionLine
SRCH_SUBSCRIPTIONPLAN	Search Subscription Plan	/app/common/search/search.nl?searchtype=Item&Item_TYPE=SubscriPlan&adv=T
SRCH_SIBSIDIARY	Search Subsidiary	/app/common/search/search.nl?searchtype=Subsidiary
SRCH_SYSTEMEMAILTEMP LATE	Search All System Email Templates	/app/common/search/search.nl?searchtype=SystemEmailTemplate
SRCH_SYSTEMNOTE	Search System Notes	/app/common/search/search.nl?searchtype=SystemNote
SRCH_TASK	Search Tasks	/app/common/search/search.nl?searchtype=Task
SRCH_TAXDETAIL	Search Tax Detail	/app/common/search/search.nl?searchtype=TaxDetail
SRCH_TAXGROUP	Search Tax Group	/app/common/search/search.nl?searchtype=TaxGroup
SRCH_TAXITEM	Search Tax Code	/app/common/search/search.nl?searchtype=TaxItem
SRCH_TAXTYPE	Search Tax Type	/app/common/search/search.nl?searchtype=TaxType
SRCH_TERMINATIONREAS ON	Search Termination Reason	/app/common/search/search.nl?searchtype=TerminationReason
SRCH_TIME	Search Track Time	/app/common/search/search.nl?searchtype=Time
SRCH_TIMEENTRY	Search Time Entry	/app/common/search/search.nl?searchtype=Timeentry
SRCH_TIMEOFFCHANGE	Search Time-Off Changes	/app/common/search/search.nl?searchtype=TimeOffChange
SRCH_TIMEOFFPLAN	Search Time-Off Plans	/app/common/search/search.nl?searchtype=TimeOffPlan
SRCH_TIMEOFFREQUEST	Time-Off Requests	/app/common/search/search.nl?searchtype=TimeOffRequest
SRCH_TIMEOFFTYPE	Search Time-Off Types	/app/common/search/search.nl?searchtype=TimeOffType

Task ID	Page Label in NetSuite	URL
SRCH_TIMESHEET	Search Timesheet	/app/common/search/search.nl?searchtype=Timesheet
SRCH_TIMESHEETAPPROVAL	Search Timesheet Approval	/app/common/search/search.nl?searchtype=TimesheetApproval
SRCH_TOPIC	Search Topic	/app/common/search/search.nl?searchtype=Topic
SRCH_TRANNUMBERAUDITLOG	Search Transaction Numbering Audit Log	/app/common/search/search.nl?searchtype=TranNumberAuditLog
SRCH_TRANSACTION	Search Transactions	/app/common/search/search.nl?searchtype=Transaction
SRCH_TRAN_BINTRNFR	Search Bin Transfers	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=BinTrnfr
SRCH_TRAN_BINWKSHT	Search Bin Putaway Worksheets	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=BinWksht
SRCH_TRAN_BLANKORD	Search Blanket Purchase Orders	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=BlankOrd
SRCH_TRAN_BOOKJOURNAL	Search Accounting Book Journal Entries	/app/common/search/search.nl?searchtype=AccountingTransaction&AccountingTransaction_TYPE=Journal&AccountingTransaction_BOOKSPECIFICTRANSACTION=T
SRCH_TRAN_BUILD	Search Assembly Builds	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Build
SRCH_TRAN_CARDCHRG	Search Credit Card Charges	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CardChrg
SRCH_TRAN_CASHRFND	Search Cash Refunds	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CashRfnd
SRCH_TRAN_CASHSALE	Search Cash Sales	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CashSale
SRCH_TRAN_CHECK	Search Checks	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Check
SRCH_TRAN_COMMISSN	Search Individual Employee Commissions	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Commissn
SRCH_TRAN_CUSTCHRG	Search Statement Charges	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustChrg

Task ID	Page Label in NetSuite	URL
SRCH_TRAN_CUSTCRED	Search Credit Memos	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustCred
SRCH_TRAN_CUSTDEP	Search Customer Deposits	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustDep
SRCH_TRAN_CUSTINVC	Search Invoices	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustInvc
SRCH_TRAN_CUSTPYMT	Search Customer Payments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustPymt
SRCH_TRAN_CUSTRFND	Search Customer Refunds	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=CustRfnd
SRCH_TRAN_DEPAPPL	Search Deposit Applications	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=DepAppl
SRCH_TRAN_DEPOSIT	Search Deposits	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Deposit
SRCH_TRAN_ESTIMATE	Search Estimates	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Estimate
SRCH_TRAN_EXPREPT	Search Expense Reports	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=ExpRept
SRCH_TRAN_FFTREQ	Search Fulfillment Requests	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=FftReq
SRCH_TRAN_FXREVAL	Search Currency Revaluations	/app/accounting/transactions/transactionsearchswitch.nl?searchtype=AccountingTransaction&AccountingTransaction_TYPE=FxReval
SRCH_TRAN_INVADJST	Search Inventory Adjustments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvAdjst
SRCH_TRAN_INVCOUNT	Search Inventory Counts	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvCount
SRCH_TRAN_INVDISTR	Search Inventory Distributions	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvDistr
SRCH_TRAN_INVREVAL	Search Inventory Cost Revaluations	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvReval

Task ID	Page Label in NetSuite	URL
SRCH_TRAN_INVTRNFR	Search Inventory Transfers	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvTrnfr
SRCH_TRAN_INVVKSHT	Search Inventory Worksheets	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=InvWksht
SRCH_TRAN_ITEMRCPT	Search Item Receipts	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=ItemRcpt
SRCH_TRAN_ITEMSHIP	Search Item Fulfillments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=ItemShip
SRCH_TRAN_JOURNAL	Search Journal Entries	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Journal
SRCH_TRAN_LIABPYMT	Search Liability Payments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=LiabPymt
SRCH_TRAN_OPPRTNTY	Search Opportunities	/app/common/search/search.nl?searchtype=Opprtny
SRCH_TRAN_PAYCHECK	Search Paychecks	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Paycheck
SRCH_TRAN_PCHKJRNL	Search Paycheck Journals	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=PChkJrnlnl
SRCH_TRAN_PURCHCON	Search Purchase Contracts	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=PurchCon
SRCH_TRAN_PURCHORD	Search Purchase Orders	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=PurchOrd
SRCH_TRAN_PURCHREQ	Search Requisitions	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=PurchReq
SRCH_TRAN_REVARRNG	Search Revenue Arrangements	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=RevArrng
SRCH_TRAN_REVCOMM	Search Revenue Commitments	/app/accounting/transactions/transactionsearchswitch.nl?searchtype=AccountingTransaction&AccountingTransaction_TYPE=RevComm
SRCH_TRAN_REVCOMRV	Search View Revenue Commitment Reversals	/app/accounting/transactions/transactionsearchswitch.nl?searchtype=AccountingTransaction&AccountingTransaction_TYPE=RevComRv

Task ID	Page Label in NetSuite	URL
SRCH_TRAN_REVCONTR	Search View Revenue Contracts	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=RevContr
SRCH_TRAN_RFQ	Search Requests For Quote	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Rfq
SRCH_TRAN_RTNAUTH	Search Return Authorizations	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=RtnAuth
SRCH_TRAN_SALESORD	Search Sales Orders	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=SalesOrd
SRCH_TRAN_STATJOURNAL	Search Statistical Journals	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Journal&Transaction_STATISTICAL=T
SRCH_TRAN_STPICKUP	Search Store Pickup Fulfillments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=StPickUp
SRCH_TRAN_TAXLIAB	Search Tax Liabilities	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=TaxLiab
SRCH_TRAN_TAXPYMT	Search Tax Payments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=TaxPymt
SRCH_TRAN_TEGPYBL	Search Issued Tegatas	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=TegPybl
SRCH_TRAN_TEGRCVBL	Search Received Tegatas	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=TegRcvbl
SRCH_TRAN_TRANSFER	Search Bank Transfers	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Transfer
SRCH_TRAN_TRNFRORD	Search Transfer Orders	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=TrnfrOrd
SRCH_TRAN_UNBUILD	Search Assembly Unbuilds	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=Unbuild
SRCH_TRAN_VENDAUTH	Search Vendor Return Authorizations	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=VendAuth
SRCH_TRAN_VENDBILL	Search Bills	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=VendBill

Task ID	Page Label in NetSuite	URL
SRCH_TRAN_VENDCRED	Search Bill Credits	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=VendCred
SRCH_TRAN_VENDPYMT	Search Bill Payments	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=VendPymt
SRCH_TRAN_VENDRFQ	Search Vendor Requests For Quote	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=VendRfq
SRCH_TRAN_WOCLOSE	Search Work Order Closes	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=WOClose
SRCH_TRAN_WOCOMPL	Search Work Order Completions	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=WOCompl
SRCH_TRAN_WOISSUE	Search Work Order Issues	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=WOIssue
SRCH_TRAN_WORKORD	Search Work Orders	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=WorkOrd
SRCH_TRAN_YTDADJST	Search Payroll Adjustment	/app/common/search/search.nl?searchtype=Transaction&Transaction_TYPE=YtdAdjst
SRCH_TWOFACTORDEVICE	Two-Factor Tokens	/app/common/search/search.nl?searchtype=TwoFactorDevice
SRCH_UNBILLORDERS	Customers by Unbilled Orders	/app/common/search/searchresults.nl?searchid=-74
SRCH_UNITSTYPE	Search Unit of Measure	/app/common/search/search.nl?searchtype=UnitsType
SRCH_USAGE	Search Usage	/app/common/search/search.nl?searchtype=Usage
SRCH_USERNOTE	Search User Notes	/app/common/search/search.nl?searchtype=UserNote
SRCH_VENDOR	Search Vendor	/app/common/search/search.nl?searchtype=Vendor
SRCH_WEEKLYTIMESHEET	Search Weekly Time Tracking	/app/common/search/search.nl?searchtype=WeeklyTimeSheet
SRCH_WORKCALENDAR	Search Work Calendar	/app/common/search/search.nl?searchtype=WorkCalendar
SRCH_WORKFLOW	Search Workflow	/app/common/search/search.nl?searchtype=Workflow
SRCH_WORKFLOWINSTANCE	Search Workflow Instances	/app/common/search/search.nl?searchtype=WorkflowInstance



Task ID	Page Label in NetSuite	URL
SRCH_WORKPLACE	Search Workplace	/app/common/search/search.nl?searchtype=Workplace
SUPT_ALLOW_LOGIN	NetSuite Support Login	/app/crm/support/allowssupportlogin.nl
SUPT_CENTER_ROLE	NetSuite Support Center	/app/login/dashboard.nl?id=
TRAN_ACTIVITY	Activity	/app/crm/calendar/activity.nl
TRAN_ADDCONTENT	Add Content	/app/center/setup/addcontent.nl
TRAN_ADDSHORTCUT	Short Cuts	/core/pages/addShortcut.nl
TRAN_ALLOCATEPAYCHECKSTOJOBS	Allocate Paycheck Expenses to Jobs	/app/accounting/transactions/allocatepaycheckstojobs.nl
TRAN_ALLOCATEREVENUEARRANGEMENTS	Allocate Revenue Arrangements	/app/accounting/revrec/allocaterevenuearrangements.nl
TRAN_ALLOCATEREVENUEARRANGEMENTS_STATUS	Allocate Revenue Arrangements Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=ALLOCATEREVARRANGEMENT&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_ALLOCATIONBATCH	Allocation Batch Status	/app/accounting/transactions/allocationbatchstatus.nl
TRAN_APPROVAL_EXPREPT	Approve Expense Reports	/app/accounting/transactions/approveexpensereport.nl
TRAN_APPROVAL_PURCHORD	Approve Purchase Requests	/app/accounting/transactions/approval.nl?type=purchord&label=Purchase%20Request
TRAN_APPROVECOMMISSN	Approve Employee Commissions	/app/accounting/transactions/approvecommissn.nl
TRAN_APPROVEPARTNERCOMMISSN	Approve Partner Commissions	/app/accounting/transactions/approvepartnercommissn.nl
TRAN_AUDIT	View Audit Trail	/app/accounting/transactions/audit.nl
TRAN_BALANCELCGACCOUNTS	Balance Location Costing Group Accounts	/app/accounting/inventory/balancelgaccounts.nl
TRAN_BALANCELCGACCOUNTS_STATUS	Balance Location Costing Group Accounts Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BALANCELCGACCOUNTS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BANKRECON	Find Matching Transactions	/app/accounting/transactions/bankrecon.nl
TRAN_BANKVIEW	Online Banking Statement	/app/accounting/transactions/bankview.nl
TRAN_BAS	Business Activity Statement	/app/accounting/reports/intl/bas.nl
TRAN_BATCHCHECK	Batch Check	/app/payroll/batchcheck.nl
TRAN_BILLOFMATERIALSINQUIRY	Bill of Materials Inquiry	/app/accounting/transactions/inventory/billofmaterialsinquiry.nl



Task ID	Page Label in NetSuite	URL
TRAN_BILLPAY_LOG	Job Status	/app/external/xml/upload/uploadlog.nl?displayType=BILLPAY
TRAN_BILLRUN	Process Billing Operations	/app/accounting/transactions/billingworkcenter/billrun.nl
TRAN_BILLRUNRESULT	View Billing Operations	/app/accounting/transactions/billingworkcenter/billrunresults.nl
TRAN_BLANKORDAPPRV	Approve Blanket Purchase Orders	/app/accounting/transactions/transactionapproval.nl?type=BlankOrd
TRAN_BUDGET	Set Up Budgets	/app/accounting/transactions/budgets.nl
TRAN_BULKAUTHCOMMIS SN_LOG	Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKAUTHCOMMISSN&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKAUTHPARTNER COMMISSN_LOG	Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKAUTHPARTNERCOMMISSN&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKBILL_LOG	Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BILLSALESORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKCOMMITREVEN UE_LOG	Job Status	/app/external/xml/upload/uploadlog.nl?displayType=BULKCOMMITREVENUE
TRAN_BULKFULFILL_LOG	Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=FULFILLSALESORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKINVOICE_LOG	Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=INVOICEBILLABLECUSTOMERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKITEMSHIPSTAT USPACK_LOG	Mark Orders Packed Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKITEMSHIPSTATUSPACK&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKITEMSHIPSTAT USSHIP_LOG	Mark Orders Shipped Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKITEMSHIPSTATUSSHIP&BulkProcSubmission_CREATEDDATE=TODAY

Task ID	Page Label in NetSuite	URL
TRAN_BULKRECEIVE_LOG	Process Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocType=BULKRECEIVEORDER&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_BULKREVENUECONTRACT_LOG	Job Status	/app/external/xml/upload/uploadlog.nl?displayType=BULKREVENUECONTRACT
TRAN_CALCULATEITEMDEMANDPLAN	Calculate Item Demand Plan	/app/accounting/inventory/demandplanning/calculateitemdemandplan.nl
TRAN_CALCULATEITEMDEMANDPLAN_STATUS	Calculate Item Demand Plan Status	/app/accounting/inventory/demandplanning/calculateitemdemandplanstatus.nl
TRAN_CALENDARPREFERENCE_NCE	Calendar Preferences	/app/crm/calendar/calendarpreference.nl
TRAN_CAMPAIGNSETUP	Marketing Preferences	/app/setup/campaignsetup.nl
TRAN_CHARGEMANAGER	Manage Charge Stages	/app/accounting/transactions/billing/chargemanager.nl
TRAN_CHECKITEMAVAILABILITY	Check Item Availability	/app/accounting/inventory/atp/checkitemavailability.nl
TRAN_CLEARHOLD	Manage Payment Holds	/app/accounting/transactions/salesordermanager.nl?type=clearholds
TRAN_CLOSEWORKORDERS_LOG	Close Work Orders Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocType=CLOSEWORKORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_COMPLETEWORKORDERS_LOG	Enter Completions Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocType=COMPLETEWORKORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_COMPONENTWHEREUSEDINQUIRY	Component Where Used Inquiry	/app/accounting/transactions/manufacturing/componentwhereusedinquiry.nl
TRAN_COPY_BUDGET	Copy Budgets	/app/accounting/transactions/copybudget.nl
TRAN_COSTEDBOMINQUIRY	Costed Bill of Materials Inquiry	/app/accounting/transactions/manufacturing/costedbillofmaterialsinquiry.nl
TRAN_CREATEAMORTIZATIONJE	Create Amortization Journal Entries	/app/accounting/transactions/createamortizationje.nl
TRAN_CREATEAMORTIZATIONJE_LOG	Create Journal Entries Status	/app/accounting/transactions/createrecognitionjestatus.nl?type=AMORTIZATION

Task ID	Page Label in NetSuite	URL
TRAN_CREATEDEGROSSJE	Create Deferred Revenue Reclassification Journal Entries	/app/accounting/transactions/createdegrossje.nl
TRAN_CREATEDEGROSSJE_LOG	Create Reclassification Journal Entries Status	/app/accounting/transactions/createdegrossjestatus.nl
TRAN_CREATEINTERCOADJJE	Create Intercompany Adjustment Journal Entries	/app/accounting/transactions/intercoadj/createintercoadjje.nl
TRAN_CREATEINTERCOADJJE_LOG	Create Journal Entries Status	/app/accounting/transactions/intercoadj/createintercoadjjestatus.nl
TRAN_CREATENEXTGENRECLASSJE	Create Deferred Revenue Reclassification Journal Entries	/app/accounting/transactions/createdegrossje.nl
TRAN_CREATENEXTGENRECLASSJE_LOG	Create Reclassification Journal Entries Status	/app/accounting/transactions/createreclassjournalstatus.nl?bulkproctype=DEFERREDREVENUERECLASS&ReclassificationStatus_CREATEDDATE=TODAY
TRAN_CREATEREVRECJE	Create Revenue Recognition Journal Entries	/app/accounting/transactions/createrevrecje.nl
TRAN_CREATEREVRECJE_LOG	Create Journal Entries Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKREVREC&bulkproctype=RECOGNIZEREVENUE&bulkproctype=RECOGNIZEPLANNEDREVENUE&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_CREATE_INVCOUNT	Create Inventory Count	/app/accounting/transactions/creatinvcnt.nl
TRAN_CREATE_JOBS_FROM_ORDERS	Create Jobs From Sales Orders	/app/accounting/transactions/jobcreationmanager.nl
TRAN_CREATE_WORK_ORDERS_FOR_STOCK	Mass Create Work Orders	/app/accounting/transactions/createworkordersforstock.nl
TRAN_CREATE_WORK_ORDERS_FOR_STOCK_DEMAND_PLANNING	Mass Create Work Orders	/app/accounting/transactions/createworkordersforstock.nl
TRAN_CREATE_WORK_ORDERS_FOR_STOCK_LOG	Process Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=CREATEWORKORDERSFORSTOCK&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_CUSTCATEGORY	Translate Category	/app/common/custom/custcategory.nl
TRAN_CUSTINVCAPPRV	Approve Invoices	/app/accounting/transactions/transactionapproval.nl?type=CustInv
TRAN_CUSTOMIZEITEMLIST	Customize Item List	/app/accounting/transactions/customizeitemlist.nl
TRAN_CUSTOMIZETRANLIST	Customize Transaction List	/app/accounting/transactions/customizetranlist.nl

Task ID	Page Label in NetSuite	URL
TRAN_DEPOSITSUMMARY	Deposit Summary	/app/accounting/transactions/depositsummary.nl
TRAN_DOMAINS	Set Up Domains	/app/setup/domains.nl
TRAN_DOMAINSADV	Set Up Domains	/app/setup/domains.nl
TRAN_EMAILPWD	Change Email Password	/app/center/emailpwd.nl
TRAN_EMPLOYEESFA	Assign Reps	/app/common/entity/employeesfa.nl
TRAN_FINCHRG	Assess Finance Charges	/app/accounting/transactions/finchrg.nl
TRAN_FORECAST	Edit Sales Rep Forecast	/app/crm/sales/forecast.nl
TRAN_FXREVAL_STATUS	Revalue Open Currency Balances Status	/app/accounting/transactions/fxrevalstatus.nl
TRAN_GENERATEFISCALPERIODS	Generate Fiscal Periods	/app/setup/period/generatefiscalperiods.nl
TRAN_GENERATEITEMSUPPLYPLAN	Generate Item Supply Plan	/app/accounting/inventory/demandplanning/generateitemsupplyplan.nl
TRAN_GENERATEITEMSUPPLYPLAN_STATUS	Generate Item Supply Plan Status	/app/accounting/inventory/demandplanning/generateitemsupplyplanstatus.nl
TRAN_GENERATETAXPERIODS	Generate Tax Periods	/app/setup/period/generatetaxperiods.nl
TRAN_GIFTCERTCREATEJE	Recognize Gift Certificate Income	/app/accounting/transactions/giftcertcreateje.nl
TRAN_GLNUMSTATUS	GL Audit Numbering Status	/app/accounting/transactions/glnumbering/glnumstatus.nl
TRAN_GSTREFUND	Process GST/HST Refund	/app/accounting/transactions/gstrefund.nl
TRAN_HISTORY	History	/app/accounting/transactions/history.nl
TRAN_IMPACT	GL Impact	/app/accounting/transactions/impact.nl
TRAN_INVOICECUSTOMERS	Invoice Billable Customers	/app/accounting/transactions/invoicecustomers.nl
TRAN_ISSUEWORKORDERS_LOG	Issue Components Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=ISSUEWORKORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_ITEMGROSSREQUIREMENTS	Gross Requirements Inquiry	/app/accounting/inventory/demandplanning/itemgrossrequirements.nl
TRAN_ITEMSHIPPACK	Mark Orders Packed	/app/accounting/transactions/itemshipmanager.nl?type=pack
TRAN_ITEMSHIPSHIP	Mark Orders Shipped	/app/accounting/transactions/itemshipmanager.nl?type=ship

Task ID	Page Label in NetSuite	URL
TRAN_JOURNALAPPROVAL	Approve Journal Entries	/app/accounting/transactions/journalapproval.nl
TRAN_LASTLOGIN	My Login Audit Portlet	/app/setup/lastlogin.nl
TRAN_LOGINAUDIT	Login Audit	/app/setup/loginAudit.nl
TRAN_MANGEARRANGEMENTS	Update Revenue Arrangements	/app/accounting/revrec/managearrangementsandplans.nl?type=arrangement
TRAN_MANAGEREVENUEARRANGEMENTS	Update Revenue Arrangements and Revenue Recognition Plans	/app/accounting/bulkprocessing/managearrangementsandplansstatus.nl?bulkprocotype=MANAGEREVENUEARRANGEMENT&bulkprocotype=MANAGEREVENUEELEMENTS&bulkprocotype=DELETEREVENUEELEMENTS&bulkprocotype=MANAGEACTUALREVENUEPLANS&bulkprocotype=MANAGEFORECASTREVPLANS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_MANAGEREVENUEARRANGEMENTS_STATUS	Update Revenue Arrangements and Revenue Plans Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=MANAGEREVENUEARRANGEMENT&bulkprocotype=MANAGEREVENUEELEMENTS&bulkprocotype=DELETEREVENUEELEMENTS&bulkprocotype=MANAGEACTUALREVENUEPLANS&bulkprocotype=MANAGEFORECASTREVPLANS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_MANAGEREVENUEPLANS	Update Revenue Recognition Plans	/app/accounting/revrec/managearrangementsandplans.nl?type=plan
TRAN_MARKBUILTWORKORDERS_LOG	Mark Work Orders Built Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=MARKBUILTWORKORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_MARKVSOEDELIVERED	Mark VSOE Delivered Status	/app/accounting/transactions/revrec/markvsoedelivered.nl
TRAN_MARKVSOEDELIVERED_LOG	Mark VSOE Delivered Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=MARKVSOEDELIVERED&BulkProcSubmission_CREATEDDATE=TODAY

Task ID	Page Label in NetSuite	URL
TRAN_MERGEREVENUEARRANGEMENTS	Merge Revenue Arrangements for Linked Sources	/app/accounting/revrec/mergerevenuearrangements.nl
TRAN_MERGEREVENUEARRANGEMENTS_STATUS	Merge Revenue Arrangements for Linked Sources Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=MERGEREVENUEARRANGEMENT&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_MGRFORECAST	Edit Sales Manager Forecast	/app/crm/sales/mgrforecast.nl
TRAN_NLVENDOR	Vendor Center	/app/center/nlvendor.nl.nl
TRAN_OPENBAL	Enter Opening Balances	/app/accounting/transactions/openbal.nl
TRAN_ORDERDEMANDPLANITEMS	Order Items	/app/accounting/transactions/orderitems.nl
TRAN_ORDERITEMS	Order Items	/app/accounting/transactions/orderitems.nl
TRAN_ORDERITEMS_LOG	Order Items Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKORDERITEMS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_ORDERPURCHREQ_LOG	Order Requisitions Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKORDERREQUISITIONS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_ORDER_REALLOCATION	Commit Orders	/app/accounting/transactions/orderreallocation.nl
TRAN_ORDER_REALLOCATION_LOG	Commit Orders Job Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=ORDERREALLOCATION&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_PAYMENTS	Payments	/app/accounting/transactions/payments.nl
TRAN_PAYROLLBATCH	Payroll Batch	/app/payroll/payrollbatch.nl
TRAN_PAYROLLRUN	Create Payroll	/app/accounting/transactions/payroll/payrollrun.nl
TRAN_PAYROLLSTATUS	View Payroll Status	/app/payroll/cdstatus.nl
TRAN_PDF_F940	Annual Federal Unemployment (940)	/app/accounting/reports/taxformfederal.nl?formtype=f940
TRAN_PDF_F941	Quarterly Federal Tax Return (941)	/app/accounting/reports/taxformfederal.nl?formtype=f941
TRAN_PLANNEDSTANDARDCOSTROLLUP	Planned Standard Cost Rollup	/app/accounting/inventory/plannedstandardcostrollup.nl



Task ID	Page Label in NetSuite	URL
TRAN_PLANNEDSTANDARD COSTROLLUP_STATUS	Planned Standard Cost Rollup Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=ROLLUPITEMCOST&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_POSTVENDORBILLVA RIANCES	Post Vendor Bill Variances	/app/accounting/transactions/vendorbillvariance/postvendorbillvariances.nl
TRAN_POSTVENDORBILLVA RIANCES_STATUS	Post Vendor Bill Variances Status	/app/accounting/transactions/vendorbillvariance/postvendorbillvariancesstatus.nl
TRAN_PREVIEWW2	Form W-2 Preview	/app/accounting/reports/w2preview.nl
TRAN_PRINT	Print Checks and Forms	/app/accounting/print/print.nl
TRAN_PRINT1096	Form 1096	/app/accounting/reports/NLPrint1096s.nl?mode=frame
TRAN_PRINT1099	Form 1099-MISC	/app/accounting/reports/NLPrint1099s.nl?mode=frame
TRAN_PRINTBARCODES	Generate Barcodes	/app/accounting/print/printbarcodes.nl?printtype=null&method=print
TRAN_PRINTMAILINGLABELS	Print Mailing Labels	/app/accounting/print/printmailinglabels.nl?printtype=null&method=print
TRAN_PRINTPRICELIST	Individual Price List	/app/accounting/print/printpricelist.nl
TRAN_PRINTSTATEMENT	Individual Statement	/app/accounting/print/printstatement.nl
TRAN_PRINTW2	Form W-2	/app/accounting/reports/NLPrintW2s.nl?mode=frame
TRAN_PRINTW2AUDIT	W-2 and 1099 Audit Information	/app/accounting/reports/printw2audit.nl?mode=frame
TRAN_PRINTW3	Form W-3	/app/accounting/reports/NLPrintW3s.nl?mode=frame
TRAN_PRINT_BOM	Print Bills of Materials	/app/accounting/print/printframe.nl?trantype=workord&printtype=bom&method=print
TRAN_PRINT_CASHSALE	Print Receipts	/app/accounting/print/printframe.nl?trantype=cashsale&printtype=transaction&method=print
TRAN_PRINT_CHECK	Print Checks	/app/accounting/print/printframe.nl?trantype=check&printtype=transaction&method=print
TRAN_PRINT_COMMERCIALINVOICE	Print Commerical Invoice	/app/accounting/print/commercialinvoice.nl
TRAN_PRINT_CUSTCRED	Print Credits Memos	/app/accounting/print/printframe.nl?trantype=custcred&printtype=transaction&method=print

Task ID	Page Label in NetSuite	URL
TRAN_PRINT_CUSTINVC	Print Invoices	/app/accounting/print/printframe.nl?trantype=custinv&printtype=transaction&method=print
TRAN_PRINT_ESTIMATE	Print Estimates	/app/accounting/print/printframe.nl?trantype=estimate&printtype=transaction&method=print
TRAN_PRINT_INTEGRATEDSHIPPINGLABEL	Print Integrated Shipping Labels	/app/accounting/print/printlabels.nl?printtype=integratedshippinglabel&method=print&title=Integrated Shipping Labels
TRAN_PRINT_ITEM_DETAIL_STATEMENT	Generate Item Detail Statements	/app/accounting/print/printframe.nl?trantype=&printtype=itemdetailstatement&method=print
TRAN_PRINT_ONE_ITEM_DETAIL_STATEMENT	Individual Item Detail Statement	/app/accounting/print/printitemdetailstatement.nl
TRAN_PRINT_PACKINGSLIP	Print Packing Slips	/app/accounting/print/printframe.nl?trantype=&printtype=packingslip&method=print
TRAN_PRINT_PAYCHECK	Print Paychecks	/app/accounting/print/printframe.nl?trantype=paycheck&printtype=transaction&method=print
TRAN_PRINT_PAYMENTVOUCHER	Print Payment Vouchers	/app/accounting/print/printframe.nl?trantype=vendpymt&printtype=paymentvoucher&method=print
TRAN_PRINT_PICKINGTICKET	Print Picking Tickets	/app/accounting/print/printframe.nl?trantype=salesord&printtype=pickingticket&method=print
TRAN_PRINT_PRICELIST	Generate Price Lists	/app/accounting/print/printframe.nl?trantype=&printtype=pricelist&method=print
TRAN_PRINT_PURCHORD	Print Purchase Orders	/app/accounting/print/printframe.nl?trantype=purchord&printtype=transaction&method=print
TRAN_PRINT_RTNAUTH	Return Authorizations	/app/accounting/print/printform.nl?printtype=transaction&trantype=rtnauth&method=print&title=Return%20Authorizations
TRAN_PRINT_SALESORD	Print Sales Orders	/app/accounting/print/printframe.nl?trantype=salesord&printtype=transaction&method=print
TRAN_PRINT_SHIPPINGLABEL	Print Shipping Labels	/app/accounting/print/printframe.nl?trantype=&printtype=shippinglabel&method=print
TRAN_PRINT_STATEMENT	Generate Statements	/app/accounting/print/printframe.nl?trantype=&printtype=statement&method=print



Task ID	Page Label in NetSuite	URL
TRAN_PROCESSCOMMISSN	Authorize Employee Commissions	/app/accounting/transactions/processcommissn.nl
TRAN_PROCESSICRTNAUTHS	Manage Intercompany Return Authorizations	/app/accounting/transactions/interco/rtnauthqueue.nl
TRAN_PROCESSICSALESORDERS	Manage Intercompany Sales Orders	/app/accounting/transactions/interco/salesordqueue.nl
TRAN_PROCESSORDER	Process Individual Order	/app/accounting/transactions/processorder.nl
TRAN_PROCESSPARTNERCOMMISSN	Authorize Partner Commissions	/app/accounting/transactions/processpartnercommissn.nl
TRAN_PURCHCONAPPRV	Approve Purchase Contracts	/app/accounting/transactions/transactionapproval.nl?type=PurchCon
TRAN_PURCHORDPROC	Bill Purchase Orders	/app/accounting/transactions/purchordermanager.nl?type=proc
TRAN_PURCHORDRECEIVE	Receive Orders	/app/accounting/transactions/purchordermanager.nl?type=receive
TRAN_PURCHREQAPPRV	Approve Requisitions	/app/accounting/transactions/transactionapproval.nl?type=PurchReq
TRAN_QUOTA	Establish Quotas	/app/crm/sales/quota.nl
TRAN_REALLOCITEMS	Reallocate Items	/app/accounting/transactions/reallocitems.nl
TRAN_RECALCULATEREVENUEFORECASTPLANS	Recalculate Revenue Forecast Plans	/app/accounting/revrec/recalculaterevenueforecastplans.nl
TRAN_RECALCULATEREVENUEFORECASTPLANS_LOG	Recalculate Revenue Forecast Plans Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=RECALCREVFORECASTPLANS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_RECONCILE	Reconcile Bank Statement	/app/accounting/transactions/reconcile.nl
TRAN_RECONCILE_CC	Reconcile Credit Card Statement	/app/accounting/transactions/reconcile.nl?page_type=cc
TRAN_REIMBURSEMENTS	Reimbursements	/app/accounting/transactions/reimbursements.nl
TRAN_Reminders	Setup Reminders	/app/center/setup/reminders.nl
TRAN_ReplenishInventory_LOG	Replenish Location By Transfer Order Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=REPLENISHINVENTORY&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_RevalueStandardCostInventory	Revalue Standard Cost Inventory	/app/accounting/inventory/revaluestandardcostinventory.nl
TRAN_RevalueStandardCostInventory_Status	Revalue Standard Cost Inventory Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=



Task ID	Page Label in NetSuite	URL
		REVALUESTDCOSTINVENTORY&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_REVARRNGAPPRV	Approve Revenue Arrangements	/app/accounting/transactions/transactionapproval.nl?type=RevArrng
TRAN_REVIEWNEGATIVEIN VENTORY	Review Negative Inventory	/app/accounting/transactions/inventory/reviewnegativeinventory.nl
TRAN_REVRECCREATEJE	Revenue Recognition Schedules	/app/accounting/transactions/revreccreateje.nl
TRAN_RTNAUTHAPPRV	Approve Return Authorizations	/app/accounting/transactions/returnauthmanager.nl?type=apprv
TRAN_RTNAUTHCREDIT	Refund Returns	/app/accounting/transactions/returnauthmanager.nl?type=credit
TRAN_RTNAUTHRECEIVE	Receive Returned Order	/app/accounting/transactions/returnauthmanager.nl?type=receive
TRAN_RTNAUTHREVERSER EVCOMMITMENT	Generate Revenue Commitment Reversals	/app/accounting/transactions/returnauthmanager.nl?type=reverserevcommitment
TRAN_SALESORDAPPRV	Approve Sales Orders	/app/accounting/transactions/salesordermanager.nl?type=apprv
TRAN_SALESORDCOMMITLE VENUE	Generate Revenue Commitments	/app/accounting/transactions/salesordermanager.nl?type=commitrevenue
TRAN_SALESORDFULFILL	Fulfill Orders	/app/accounting/transactions/salesordermanager.nl?type=fulfill
TRAN_SALESORDPROC	Invoice Sales Orders	/app/accounting/transactions/salesordermanager.nl?type=proc
TRAN_SALESORDREVENUE CONTRACT	Generate Single Order Revenue Contracts	/app/accounting/transactions/salesordermanager.nl?type=createrevenuecontracts
TRAN_SAVEDASHBOARD	Publish Dashboard	/app/center/setup/savedashboard.nl
TRAN_SEARCH	Search	/app/common/search/search.nl
TRAN_SHORTCUTS	Add Shortcuts	/app/center/shortcuts.nl
TRAN_SNAPSHOTCOMPOS ER	Custom Snapshot Report	/app/reporting/snapshotcomposer.nl
TRAN_SNAPSHOTS	Setup Snapshots	/app/center/setup/snapshots.nl
TRAN_STATSCHEDULE	Statistical Schedule Status	/app/accounting/transactions/statistical/statisticschedulestatus.nl
TRAN_STPICKUP	Manage Store Pickup	/app/accounting/transactions/transactionlist.nl?Transaction_TYPE=StPickUp
TRAN_TAXPERIODS	Generate Tax Reporting Periods	/app/setup/period/generatetaxperiods.nl

Task ID	Page Label in NetSuite	URL
TRAN_TIMEAPPROVAL	Approve Time	/app/accounting/transactions/timeapproval.nl
TRAN_TIMEBILL	Track Time	/app/accounting/transactions/timebill.nl
TRAN_TIMEBILL_WEEKLY	Weekly Time Sheet	/app/accounting/transactions/timebill.nl?weekly=T
TRAN_TIMECALC	Calculate Time	/core/pages/timecalc.nl
TRAN_TIMEENTRYAPPROVAL	Approve Time Entry	/app/accounting/transactions/timeentryapproval.nl
TRAN_TIMEPOST	Post Time	/app/accounting/transactions/timepost.nl
TRAN_TIMER	Timer	/core/pages/timer.nl
TRAN_TIMESHEETAPPROVAL	Approve Time	/app/accounting/transactions/timesheetapproval.nl
TRAN_TIMEVOID	Void Time	/app/accounting/transactions/timevoid.nl
TRAN_TRNFRORDAPPRV	Approve Transfer Orders	/app/accounting/transactions/transferordermanager.nl?type=apprv
TRAN_UPDATEREVENUERECOGNITIONPLANS	Edit Revenue Recognition Plans	/app/accounting/revrec/updaterevenuerecognitionplans.nl
TRAN_UPDATEREVENUERECOGNITIONPLANS_LOG	Update Revenue Recognition Plans Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=UPDATEREVRECPLANS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_USERPREFS	Set Preferences	/app/center/userprefs.nl
TRAN_VENDAUTHAPPRV	Approve Vendor Returns	/app/accounting/transactions/vendauthmanager.nl?type=apprv
TRAN_VENDAUTHCREDIT	Credit Vendor Returns	/app/accounting/transactions/vendauthmanager.nl?type=credit
TRAN_VENDAUTHRETURN	Ship Vendor Returns	/app/accounting/transactions/vendauthmanager.nl?type=return
TRAN_VENDBILLAPPRV	Approve Bills	/app/accounting/transactions/vendorbillmanager.nl?type=apprv
TRAN_VENDBILLPURCHORD	New Purchase Order	/app/accounting/transactions/venbillpurchord.nl
TRAN_VENDPYMTS	Pay Bills	/app/accounting/transactions/vendpymts.nl
TRAN_VENDPYMT_LOG	Pay Bills Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=BULKPAYBILLS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_WITHDRAWINVENTORY_LOG	Withdraw By Transfer Order Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkproctype=WITHDRAWINVENTORY&

Task ID	Page Label in NetSuite	URL
		BulkProcSubmission_CREATEDDATE=TODAY
TRAN_WORKORDBUILD	Build Work Orders	/app/accounting/transactions/salesordermanager.nl?type=build
TRAN_WORKORDBUILD_LOG	Build Work Orders Status	/app/accounting/bulkprocessing/bulkprocessingstatus.nl?bulkprocotype=BUILDWORKORDERS&BulkProcSubmission_CREATEDDATE=TODAY
TRAN_WORKORDCLOSE	Close Work Orders	/app/accounting/transactions/workordermanager.nl?type=close
TRAN_WORKORDCOMPLETE	Enter Completions	/app/accounting/transactions/workordermanager.nl?type=complete
TRAN_WORKORDISSUE	Issue Components	/app/accounting/transactions/workordermanager.nl?type=issue
TRAN_WORKORDMARKBUILT	Mark Work Orders Built	/app/accounting/transactions/workordermanager.nl?type=markbuilt
TRAN_WORKORDMARKFIRMED	Mark Work Orders Firmed	/app/accounting/transactions/workordermanager.nl?type=markfirmed
TRAN_WORKORDMARKRELASED	Mark Work Orders Released	/app/accounting/transactions/workordermanager.nl?type=markreleased



# SuiteScript Governance



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.



**Important:** SuiteScript thresholds are based on the volume of activity that a company's users can manually generate. However, automated functions that generate excessive levels of activity may trigger metering of script execution as referenced in the [NetSuite Main Terms of Service \(TOS\)](#).

To optimize application performance, NetSuite has implemented a SuiteScript governance model based on usage units. If the number of allowable units is exceeded, the script is terminated.

Usage units are tracked on two levels: the **API level** and the **script type level**. Each SuiteScript API consumes a system-defined number of processing units, and each script type can execute a system-defined number of units.

See these topics to learn about unit governance as it applies to individual APIs and specific **script types**:

- [API Governance](#)
- [Script Usage Unit Limits](#)

Note that NetSuite also governs the amount of logging that can be done by a company in any 60 minute time period. For complete details, see [Governance on Script Logging](#).



**Note:** NetSuite has put internal mechanisms in place to detect "runaway scripts" that include infinite loops. When they are caught, these scripts will be terminated and an `SSS_INSTRUCTION_COUNT_EXCEEDED` error message is thrown. Should you receive this error, NetSuite recommends that you examine the `for` loops in your script to ensure that they contain either a terminating condition or a condition that can be met.

## API Governance

The following table lists each API and the units each consumes. Notice the APIs marked with an asterisk consume a different number of units based on the type of record they are running on. This kind of governance model takes into account the NetSuite processing requirements for three categories of records: custom records, standard transaction records, and standard non-transaction records.

Custom records, for example, require less processing than standard records. Therefore, the unit cost for custom records is lower to be commensurate with the processing required for standard records. Similarly, standard non-transaction records require less processing than standard transaction records. Therefore, the unit cost for standard non-transaction records is lower to be commensurate with the processing required for standard transaction records.



**Note:** Standard **transaction** records include records such as Cash Refund, Customer Deposit, and Item Fulfillment. Standard **non-transaction** records include records such as Activity, Inventory Item, and Customer. In the section on [SuiteScript Supported Records](#) in the NetSuite Help Center, see the “Record Category” column. All record types not categorized as **Transaction** are considered to be standard non-transaction records. Custom List and Custom Record are considered to be custom records.

API	Unit Usage per API	Example
nlapiDeleteFile nlapiInitiateWorkflow nlapiTriggerWorkflow nlapiScheduleScript nlapiSubmitConfiguration nlapiSubmitFile nlobjEmailMerger.merge()	20	A user event script on a standard transaction record type (such as Invoice) that includes one call to nlapiDeleteRecord and one call to nlapiSubmitRecord -- consumes 40 units (assuming no other nlapi calls were made). In this case, the user event script consumes 40 units out of a possible 1,000 units available to user event scripts. (See <a href="#">Script Usage Unit Limits</a> for the total units allowed for a user event script.)
nlapiDeleteRecord nlapiSubmitRecord	When used on standard transactions: 20 When used on standard non-transactions: 10 When used on custom records: 4	
nlapiAttachRecord nlapiDetachRecord nlapiExchangeRate nlapiGetLogin nlapiLoadConfiguration nlapiLoadFile nlapiMergeRecord nlapiRequestURL nlapiRequestURLWithCredentials nlapiSearchGlobal nlapiSearchRecord nlapiSendCampaignEmail nlapiSendEmail nlapiVoidTransaction nlapiXMLToPDF	10	A scheduled script on a standard non-transaction record type (such as Customer) that includes one call to nlapiLoadRecord, one call to nlapiTransformRecord, one call to nlapiMergeRecord, and one call to nlapiSendEmail consumes 30 units. In this case, the scheduled script consumes 30 units out of a possible 10,000 units available to scheduled scripts. (See <a href="#">Script Usage Unit Limits</a> for the total units allowed for a scheduled script.)
nlobjSearchResultSet.getResults nlobjSearchResultSet. forEachResult		

API	Unit Usage per API	Example
nlapiCreateRecord nlapiCopyRecord nlapiLookupField nlapiLoadRecord nlapiSubmitField nlapiTransformRecord	When used on standard transactions: 10 When used on standard non-transactions: 5 When used on custom records: 2	<p><b>Note:</b> To conserve units, only use nlapiSubmitField on fields that are available through inline edit. For more information, see <a href="#">Updating a field that is available through inline edit</a> and <a href="#">Updating a field that is not available through inline edit</a>.</p>
nlapiLoadSearch nlobjJobManager.getFuture nlobjSearch.saveSearch	5	
nlapiLogExecution	See <a href="#">Governance on Script Logging</a> .	
nlapiSetRecoveryPoint nlapiSubmitCSVImport nlobjJobManager.submit	100	
All other SuiteScript APIs	0	

## Script Usage Unit Limits

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The following table lists the maximum units allowed for a particular script type. You can use `nlobjGetContext.getRemainingUsage()` to see how many units you have remaining for a particular scheduled, user event, portlet, client, or Suitelet script.

Script Type	Total Units Allowed per Script	Notes
Scheduled Scripts	10,000	A scheduled script that includes two calls to <code>nlapiTransformRecord</code> , one call to <code>nlapiMergeRecord</code> , and one call to <code>nlapiSendEmail</code> consumes 40 units out of a possible 10,000 available.
User Event Scripts	1,000	<p><b>Note:</b> Scheduled scripts with potentially long execution times should include an <code>nlapiYieldScript()</code> call to avoid exceeding their allowed governance.</p>
Client Scripts	1,000	Be aware that client scripts are metered on a <b>per-script</b> basis. If an account has one <b>form-level</b> client script attached to a form, and one <b>record-level</b> client script deployed to the

Script Type	Total Units Allowed per Script	Notes
		record (which contains the form), <b>each</b> client script can total 1000 units. Usage units are not shared by all the client scripts associated with a form or record.
		<p><b>Note:</b> For information on form- and record-level client scripts, see <a href="#">Form-level and Record-level Client Scripts</a>.</p>
Suitelets	1,000	<p>For example, a Suitelet that calls <code>nlapiCreateRecord</code> and <code>nlapiRequestURL</code> consumes 20 units out of a possible 1,000 units available.</p> <p><b>Note:</b> Regardless of the 1,000 unit limit for Suitelets, developers should design their Suitelets to be responsive to users, otherwise user experience may be impacted.</p>
RESTlets	5,000	
Portlet Scripts	1,000	
Mass Update Scripts		You can have 1000 units per record/invocation of the script.
Bundle Installation Scripts	10,000	Bundle installation scripts are governed by a maximum of 10,000 units per execution.
Workflow Action Scripts  (also referred to as <a href="#">Custom Action</a> in SuiteFlow)	1,000	<p>Use workflow action scripts to create custom actions in your workflow.</p> <p>Note that within one workflow state, all actions combined cannot exceed 1000 units. Therefore, if you have developed a custom action (using a workflow action script) that consumes 990 units, be aware of the unit consumption of the other actions within that state.</p> <p>See the help topic <a href="#">Workflow Action Usage Units</a> for a list of all workflow actions and the units they consume when executed within a state.</p>

**Note:** NetSuite also enforces a usage limit of 1000 units for SSP application scripts. For more information, see the help topic [SSP Application Governance](#).

## Monitoring Script Usage

You can monitor SuiteScript unit usage through the `nlobjContext.getRemainingUsage()` method.

**Note:** To access the `getRemainingUsage()` method, call `nlapiGetContext()` to instantiate the `nlobjContext` object.

You can also monitor script units by running the script in the SuiteScript Debugger. After a script completes execution, unit governance details appear on the Execution Log tab in the SuiteScript



Debugger console. Note that emailed error messages also include a line item for **Script Usage**. Script Usage shows the number of units that were executed in the script before the error was hit.

### Example 1

This sample shows how to instantiate the nlobjContext object, and then call getRemainingUsage() so that the script's remaining usage units appear in the execution log.

```
var context = nlapiGetContext();
nlapiLogExecution('DEBUG', 'remaining usage', context.getRemainingUsage());
```

### Example 2

This sample shows how to instantiate the nlobjContext object, and then check for remaining units. If the units remaining are greater than 50, the script will execute a certain set of instructions.

```
var context = nlapiGetContext();
if (parseInt(context.getRemainingUsage()) > 50)
{
    //execute code here
}
```

## Governance on Script Logging

**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

NetSuite governs the use of [nlapiLogExecution\(type, title, details\)](#). The governance model is meant to safeguard against unreasonably excessive logging, which can negatively affect performance for other NetSuite customers sharing the same database. The governance model is not meant to impact companies (or scripts) that are using nlapiLogExecution() appropriately.

The governance model is as follows:

Within a **60 minute** time period, a company is allowed to make up to 100,000 calls to nlapiLogExecution() across **all** of their scripts.

If within a **60 minute** time period NetSuite detects a script is excessively logging (and pushing a company close to the 100,000 nlapiLogExecution() call limit), NetSuite will change the offending script's log level to the next level higher. The offending script will continue its execution, however, its log level will go from Debug to Audit, or Audit to Error, or Error to Emergency, depending on the script.

**Note:** For information on script log levels, see [Setting Script Execution Log Levels](#).

Changing log levels ensures that the offending script continues to execute and helps prevent an inordinate amount of logging from only one company.

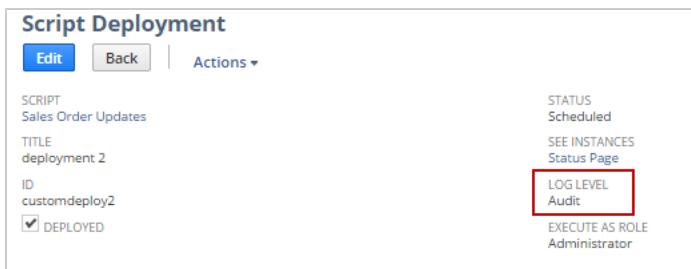
The capacity for script execution logs is shared by customers on the same database. For further protection against excessive logging, script execution logs are governed by a total storage limit on each instance of the NetSuite database. On each NetSuite server, if the database table that stores logs reaches this limit, all logs (across all customers on that server) are purged. For this reason, NetSuite recommends that you store information using custom records.



## Example

Company ABC has 10 scripts running during a **60 minute** period. If one out of the 10 scripts calls `nlapiLogExecution('DEBUG', 'My log', x.getID())` 70,000 times within only a 20 minute time period, NetSuite will raise the script's log level.

The change to the log level will appear in the **Log Level** field on the script's Script Deployment page (see figure). In the figure below, if the offending script's Log Level was originally set to Debug, NetSuite will increase the log level to Audit. This means the line of code that reads `nlapiLogExecution('DEBUG', 'My log', x.getID())` will continue to execute, however nothing will be logged, as the log level for the script has been raised to Audit.



## Script Owners Are Notified

If NetSuite detects that one script is logging excessively, the owner of the script is notified. The script owner is alerted that the script is the primary contributor to his or her company possibly exceeding the 100,000 logging threshold (for a **60 minute** time period).

NetSuite sends notifications through email and adds a log entry to the script's Execution Log. Both the email and the NetSuite-generated log alerts script owners that a script's Log Level has been increased.

## Search Result Limits

Search results are limited to 1000 records when you execute SuiteScript searches using `nlapiSearchRecord(...)`. For information on working with SuiteScript searches in NetSuite, see [Searching Overview](#) and [Search APIs](#).

Note that if you load an existing saved search using `nlapiLoadSearch(...)`, and then call `nlobjSearch.runSearch()` to return a result set of `nlobjSearchResultSet` objects, you can get up to 4000 results returned. See `nlobjSearchResultSet.forEachResult(callback)` for details.

# Multiple Shipping Routes and SuiteScript

The following topics are covered in this section. If you are unfamiliar with the Multiple Shipping Routes feature, it is recommended that you read each topic sequentially.

- What is the Multiple Shipping Routes feature?
- How does MSR work in SuiteScript?
- Which fields are associated with MSR?
- Does MSR work on existing custom forms?
- Multiple Shipping Routes Sample Code for SuiteScript
- Do I need to make code changes to existing SuiteScript code?

## What is the Multiple Shipping Routes feature?

With the Multiple Shipping Routes (MSR) feature, you can associate several items with one transaction, and then set different shipping addresses and shipping methods for each item. Transaction types such as sales order, cash sale, invoice, estimate, and item fulfillment all support MSR.

**Note:** For additional information on this feature, as well as steps for enabling MSR in your account, see the help topic [Multiple Shipping Routes](#) in the NetSuite Help Center.

The following figure shows a sales order with three items. When MSR is enabled in your account, and the *Enable Item Line Shipping* check box is selected on the transaction, each item can have its own ship-to address and shipping method. The ship-to address is specified in the *Ship To* column; the shipping method is specified in the *Ship Via* column. The SuiteScript internal IDs for each field are shown in the figure below.

OPTIONS	GIFT CERTIFICATE	SHIP TO	CARRIER	SHIP VIA	CREATE PO	ALT. SALES	EXPECTED SHIP DATE
		10 ABC Street	FedEx/More	Federal Express			
		123 X Street	FedEx/More	US Mail			

In the UI, after all items have been added to the transaction (a sales order in this example), you must then create individual shipping groups by clicking the **Calculate Shipping** button on the Shipping tab. A shipping group includes all details related to where the item is being shipped from, where it is being shipped to, the item's shipping method, and its shipping rate (see figure).

The screenshot shows the NetSuite Sales Order page with the 'Shipping' tab selected. At the top, there is a 'SHIP DATE' field set to '8/19/2014' and a 'SHIP COMPLETE' checkbox. Below this is a 'Shipment' section with 'Calculate Shipping' and 'Clear All Lines' buttons. The table below lists items grouped into two shipping categories:

SHIP FROM	SHIP TO	SHIP VIA	SHIPPING RATE	HANDLING RATE
shipping address san mateo, CA 94403 US	999 ABC Street San Mateo, WA 99403 US	Federal Express	15.00	0.00
shipping address san mateo, CA 94403 US	123 X Street San Mateo, CA 94403 US	US Mail	0.00	0.00

Annotations include a downward arrow pointing to the second row labeled 'shipgroup1' and another arrow pointing to the third row labeled 'shipgroup2'.

Although there is no UI label called "Shipping Group," SuiteScript developers can verify that shipping groups have been created by either looking in the UI after the record has first been submitted or by searching on the record and specifying *shipgroup* as one of the search return columns. See the code sample called [Get the shipping groups created for the sales order](#) for details.

The previous figure shows the UI equivalent of two separate shipping groups on a sales order. These groups are *ship group 1* and *ship group 2*.

Note that although the sales order included three items, only two shipping groups were generated. This is because the shipping information for two of the items is the same (123 Main Street for ship-to, and DHL for shipping method). The third item contains shipping details that are not like the previous two items. Therefore, this order contains three items, but only two different shipping groups.

## How does MSR work in SuiteScript?

When working with MSR-enabled transactions in SuiteScript, developers should be aware of the following:

- In SuiteScript, at the time of creating a sales order, you cannot override the default shipping rate that has been set for an item. SuiteScript developers should be aware of this when creating **user event** and **scheduled** scripts.
- There is no SuiteScript equivalent of the Calculate Shipping button that appears on the Shipping tab. In SuiteScript, shipping calculations are handled by the NetSuite backend when the transaction is submitted.
- The `nlapiTransformRecord(...)` API includes an optional *shipgroup* setting. For example:

```
var itemFulfillment = nlapiTransformRecord('salesorder', id, 'itemfulfillment', { 'shipgroup' : 50 });
var fulfillmentId = nlapiSubmitRecord(itemFulfillment, true);
```

When working with MSR-enabled transactions, you must specify a value for *shipgroup* during your transforms. If you do not specify a value, 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 a code sample that shows how to transform a sales order to an item fulfillment, see [Transform the sales order to create an item fulfillment](#).

- In both the UI and in SuiteScript, if you make any update to any item on MSR-enabled transactions, this action may result in changes to the shipping cost. Every time an item is updated and the record is submitted, NetSuite re-calculates the shipping rate. NetSuite calculates all orders based on "real-time" shipping rates.
- In both the UI and in SuiteScript, the only *transformation* workflow that is impacted by MSR is the sales order to fulfillment workflow. Invoicing and other transaction workflows are not impacted.

## Which fields are associated with MSR?

The following table lists UI field labels and their corresponding SuiteScript internal IDs for all MSR-related fields.

UI Label	Element Name	Note
Enable Item Line Shipping	ismultishipto	Set to 'T' to allow for multiple items with separate shipping address/methods
Ship To	shipaddress	References the internal ID of the customer's shipping address. You can get the internal ID by clicking the Address tab on the customer's record. The address ID appears in the ID column.
		<p> <b>Note:</b> The Show Internal ID preference must be enabled for address IDs to show.</p>
Ship Via	shipmethod	References the internal ID of the item. Go to the Items list to see all item IDs.
		<p> <b>Note:</b> The Show Internal ID preference must be enabled for address IDs to show.</p>
Shipping sublist	shipgroup	Each item that has a separate shipping address/shipping method will have a unique shipgroup number. When you transform a sales order to create a fulfillment, and the sales order has MSR enabled, you will need to specify a shipgroup value (1, 2, 3, etc) for each shipgroup you want fulfilled. See figure below.
		<p> <b>Important:</b> If no shipgroup value is specified, only the items associated with the first shipgroup (1) will be fulfilled.</p>

This figure shows two different shipping groups: ship group 1 and ship group 2. When transforming the transaction using `nlapITransformRecord(...)`, you must specify each item you want fulfilled based on its *shipgroup* value.

SHIP FROM	SHIP TO	SHIP VIA	SHIPPING RATE	HANDLING RATE
shipping address san mateo, CA 94403 US	999 ABC Street San Mateo, WA 99403 US	Federal Express	15.00	0.00
shipping address san mateo, CA 94403 US	123 X Street San Mateo, CA 94403 US	US Mail	0.00	0.00

## Does MSR work on existing custom forms?

Yes. However, after you enable Multiple Shipping Routes in your account, you must also enable MSR on your custom form by adding the Enable Line Item Shipping check box to the Items sublist. For steps on adding this check box to a custom form, see *Multiple Shipping Routes* in the NetSuite Help Center.



**Note:** After MSR is enabled in your account, the Enable Line Item Shipping check box is automatically added to the Items sublist on standard forms.

## Multiple Shipping Routes Sample Code for SuiteScript

The following samples show a typical workflow for MSR-enabled transactions. Note that these samples reference the sales order as the transaction type.

1. Create a sales order and set your items
2. Get the shipping groups created for the sales order
3. Transform the sales order to create an item fulfillment
4. Search for MSR-enabled sales orders that have not been fulfilled

### Create a sales order and set your items

```
// Create Sales order with two items and two different shipping addresses
var record = nlapiCreateRecord('salesorder');
record.setFieldValue('entity', 87); //set customer ID

// Set values for the first item
record.setLineItemValue('item', 'item', 1, 380);
record.setLineItemValue('item', 'quantity', 1, 1);
record.setLineItemValue('item', 'shipaddress', 1, 84);
record.setLineItemValue('item', 'shipmethod', 1, 37);

// Set values for the second item
```

```

record.setLineItemValue('item', 'item', 2, 440);
record.setLineItemValue('item', 'quantity', 2, 1);
record.setLineItemValue('item', 'shipaddress', 2, 275);
record.setLineItemValue('item','shipmethod', 2, 37);

var id = nlapiSubmitRecord(record, true);

```

## Get the shipping groups created for the sales order

```

var columns = new Array();
var filters = new Array();
filters[0] = new nlobjSearchFilter('internalid', null, 'is', id);
filters[1] = new nlobjSearchFilter('shipgroup', null, 'greaterthan', 0);
columns[0] = new nlobjSearchColumn('shipgroup');
var searchresults = nlapiSearchRecord('salesorder', null, filters, columns);
var shipgroups = new Array();
for( i=0; i< searchresults.length ; i++ )
{
    shipgroups[i] = searchresults[i].getValue('shipgroup');
}

```

## Transform the sales order to create an item fulfillment

```

// Transform the sales order and pass each of the shipping groups
for(i=0; i< shipgroups.length ; i ++ )
{
    var itemFulfillment = nlapiTransformRecord('salesorder', id, 'itemfulfillment', { 'shipgroup' :
        shipgroups[i] })
    var fulfillmentId = nlapiSubmitRecord(itemFulfillment, true)
}

```

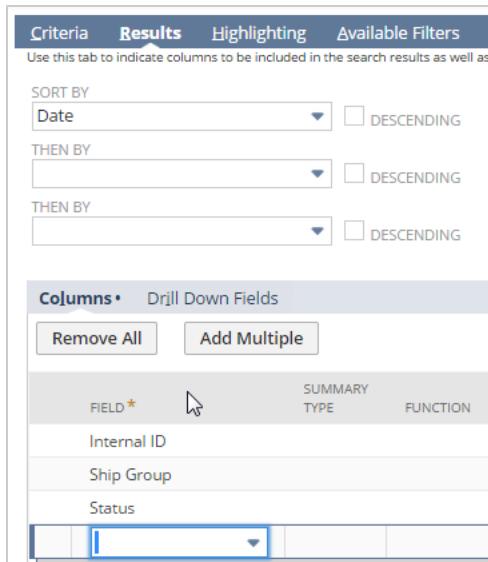


**Important:** If you do not pass a value for *shipgroup*, only the first item on the sales order is fulfilled.

## Search for MSR-enabled sales orders that have not been fulfilled

1. Create a saved search to obtain shipping group IDs. The following figures show the criteria and results column values to set.

Criteria		Results	Highlighting	Available Filters	Audience	Roles	Email	Audit Trail	Execute								
Use this tab to specify criteria that narrow down your search.																	
<input type="checkbox"/> USE EXPRESSIONS																	
<b>Standard • Summary</b> <table border="1"> <thead> <tr> <th>FILTER *</th> <th>DESCRIPTION *</th> </tr> </thead> <tbody> <tr> <td>Type</td> <td>is Sales Order</td> </tr> <tr> <td>Status</td> <td>is Sales Order:Pending Fulfillment</td> </tr> <tr> <td>Ship Group</td> <td>is greater than 0</td> </tr> </tbody> </table>										FILTER *	DESCRIPTION *	Type	is Sales Order	Status	is Sales Order:Pending Fulfillment	Ship Group	is greater than 0
FILTER *	DESCRIPTION *																
Type	is Sales Order																
Status	is Sales Order:Pending Fulfillment																
Ship Group	is greater than 0																



2. Next, run your saved search in SuiteScript to verify the same results are returned as in the saved searched performed in the UI. Then transform all unfulfilled orders based on shipgroup ID. For example,

```
var results = nlapiSearchRecord('salesorder', 17); // where 17 is the internal ID of the previous saved search
for ( var i = 0; results != null && i < results .length; i++)
{
    var id = results[i].getValue('internalid');
    var shipgroup= results[i].getValue('shipgroup');

    // Transform
    var itemFulfillment = nlapiTransformRecord('salesorder', id, 'itemfulfillment', { 'shipgroup' : shipgroup })
    var fulfillmentId = nlapiSubmitRecord(itemFulfillment)
}
```

## Do I need to make code changes to existing SuiteScript code?

Simply enabling the MSR feature in your NetSuite account does not require any code changes. However, after you enable MSR on individual transactions (by selecting the *Enable Item Line Shipping* check box on a transaction's Items sublist), you may need to make the following updates to your code:

1. When you first enable the MSR feature, you will be prompted to enable the *Per-line taxes* feature. Therefore, when you add items to a transaction that has MSR enabled, AND you want set a tax for your items, you will need to set a *taxcode* value for each line item. For example,

```
nlapiSetLineItemValue('item', 'taxcode', 1, '230' ) // where 230 is the tax code internal ID
```

Note that if you do not wish to add taxes to an item, you are not required to set a value for *taxcode*. In other words, if you want to add taxes, you must add them on a per-line basis.

If you have never set *taxcode* values on any of your transactions, and you do not wish to add *taxcode* values, no code changes are required.

2. If MSR is enabled on the transaction, you can search for the transaction, get the ID, and then fulfill the order. With MSR enabled, your existing search will now have to include a *shipgroup* column in your search.
3. Any sales order to item fulfillment transformation code will now have to include *shipgroup* as a transaction value. For example:

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

A transformation default for salesorder->itemfulfillment was added to the SuiteScript API so that the shipping route (shipgroup) can be defaulted in during transformations.



# Referencing the currencyname Field in SuiteScript



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The **currencyname** field in SuiteScript is intended to be read-only and not a submittable field (in other words, this field should not be accessible from user event scripts). For example, using the `nlapiGetFieldValue(...)` or `nlapiLookupField(...)` functions on **currencyname** will either return no value or will return an error.

To return currency information, you should instead reference the **currency** field.

If you must return currency name information, you will need to load the transaction and call `getFieldValue()`. For example,

```
nlapiLoadRecord().getFieldValue('currencyname');
```



# SuiteScript Developer Resources



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

See these links for SuiteScript developer resources:

- [SuiteScript FAQ](#)
- [NetSuite User Group](#)
- [NetSuite Developer Portal](#)

## SuiteScript and SuiteFlow Impact of Version 2014 Release 2 Address Customization Changes



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

The address customization feature to be implemented in Version 2014 Release 2 supports custom address fields and custom address forms, and moves address fields into a new address subrecord. This feature impacts all record types that contain address fields, including transactions, entities, subsidiary, company information, location, and workplace. The address form to use for each record is based on its country, so whenever a new country is selected in the address subrecord, a new address form is loaded, resetting values for all address fields.

The address customization feature is largely backwards compatible. However, this feature includes changes to SuiteScript and SuiteFlow address support that may impact a small number of existing scripts and workflows, causing unexpected results. The following sections describe these changes and suggest revisions that you can make to ensure continued correct functioning:

- [Validate Field, Field Changed, and Post Sourcing Events May Not Fire When Addresses Set](#)
- [All Address Fields Reset When Country Field Set in Dynamic Mode](#)
- [Getting and Setting Text through Parent Records Not Supported for Address Fields](#)
- [Changes Required for Code that Gets Address Field Metadata](#)
- [nlapiGetLineItemValue and nlapiSetLineItemValue Functions Not Supported for Address Fields in Dynamic Mode](#)
- [Some Set Field Workflow Actions Not Supported on Address Fields](#)



**Important:** The approach to some of these issues is to use subrecord APIs for scripting with addresses. For information about scripting with address subrecords, see [Using SuiteScript with Address Subrecords](#).

## Validate Field, Field Changed, and Post Sourcing Events May Not Fire When Addresses Set

With the new address architecture, Validate Field, Field Changed, and Post Sourcing events in client scripts deployed to some parent record types, and in custom code on these parent record type forms,

are no longer triggered by address field changes. This problem impacts scripts and custom form code for entities and for the item fulfillment transaction type. Note that scripts and custom form code for other transaction types are not impacted by this problem and should continue to work as expected. Client side scripting is not supported for subsidiary, company information, location, or workplace, so these record types are not impacted.

**Solution:**

Generally to address this problem, you can move the code, from the client event script deployed to the parent record or the custom code on the parent record type form, to the Custom Code subtab of the address form for the record type.

However, you need a different solution for code that sets values of fields that are in the parent record, because code on the address form does not have access to the parent record. You can modify this type of code within the client event script that is deployed to the parent record type, setting it to fire on a Field Changed event on the address text field that is available to the parent record. Field Changed events on address text fields fire when changes to the address subrecord are committed. The Field Changed code can load the address subrecord, check which address fields have changed, and set parent field values accordingly.

## All Address Fields Reset When Country Field Set in Dynamic Mode

Every time the country field is set in an address subrecord, the address form associated with that country is loaded. When the address form is loaded, all address field values are reset. If a script sets the country field value after other address field values are set, these other address fields may incorrectly end up with null values.

**Solution:**

All scripts running in dynamic mode, that set address field values, always need to set the country field value FIRST. If you have any scripts that set other address field values before the country field value, modify the code to set the country first.

## Getting and Setting Text through Parent Records Not Supported for Address Fields

With the new address architecture, the `nlobjRecord.getFieldText(name)` method, the `nlobjRecord.setFieldText(name, text)` method, the `nlapiGetFieldText(fldnam)` function, and the `nlapiSetFieldText(fldnam, txt, firefieldchanged, synchronous)` function are no longer supported for address fields that have become part of the address subrecord.

**Note:** These methods and functions continue to work for address fields that are part of the parent record.

**Solution:**

If you have any scripts that use these methods or functions on address fields that are part of the address subrecord rather than the parent record, modify the code to use `nlobjRecord.getFieldValue(name)`, `nlobjRecord.setFieldValue(name, value)`, `nlapiGetFieldValue(fldnam)`, or `nlapiSetValue(fldnam, value, firefieldchanged, synchronous)` instead.

## Changes Required for Code that Gets Address Field Metadata

The nlobjRecord.getLineItemField(group, fldnam, linenum) method, nlobjRecord.getField(fldnam) method, and nlapiGetField(fldnam) function are no longer supported to get address field metadata. Address fields are now part of the address subrecord, so you will need to get metadata from the subrecord rather than the parent record.

For example, the following code will return null for the country field on the address sublist:

```
var customer_record = nlapiCreateRecord('customer');
var country_field = customer_record.getLineItemField('addressbook', 'country', 1);
```

### Solution:

If you have any scripts that get address field metadata from a parent record, you need to modify the code to get metadata from the subrecord.

You can use nlapiCreateCurrentLineItemSubrecord(sublist, fldnam) to return the nlobjSubrecord object, as shown in the following example:

```
var_customer_record = nlapiCreateRecord('customer');
customer_record.selectLineItem('addressbook', 1);
var addrSubrecord = customer_record.createCurrentLineItemSubrecord('addressbook', 'addressbookaddress');
var country_field = addrSubrecord.getField('country');
```

## nlapiGetLineItemValue and nlapiSetLineItemValue Functions Not Supported for Address Fields in Dynamic Mode

The nlapiGetLineItemValue(type, fldnam, linenum) and nlapiSetLineItemValue(type, fldnam, linenum, value) functions are no longer supported for address fields in scripts running in dynamic mode.

For example, if the following code is run in dynamic mode, it will return the SSS\_INVALID\_FIELD\_ON\_SUBRECORD\_OPERATION error.

```
nlapiGetLineItemValue('addressbook', 'country', 1);
```

### Solution:

Modify the code to use nlapiGetCurrentLineItemValue( type, fldnam) or nlapiSetCurrentLineItemValue(type, fldnam, value, firefieldchanged, synchronous).

For example, instead of the above code, use code like the following:

```
nlapiSelectLineItem('addressbook', '1');
nlapiGetCurrentLineItemValue('addressbook', 'country');
```



## Address Fields on the Company Information Page are Still Accessed with nlapiLoadConfiguration

In most server-side scripts, addresses are accessed with the subrecord APIs (see [Scripting the Address Subrecord](#) for more information). Scripts that access addresses on the Company Information page are an exception to this rule. You cannot use the subrecord APIs to access these fields. You must access address fields on the Company Information page with [nlapiLoadConfiguration\(type\)](#) (in the same way you access other fields on this page).

The following example loads the Company Information page and then accesses the shipping address.

```
//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');
```

## Some Set Field Workflow Actions Not Supported on Address Fields

Due to the changes implemented by the address customization feature, the following workflow actions are no longer supported on address fields:

- Set Field Display Label
- Set Field Display Type
- Set Field Mandatory

Note the Set Field Value workflow action continues to be supported for address fields.

### Solution:

If any of your workflows currently include the unsupported actions on address fields, you need to delete these actions from the workflow and implement the behavior through address form customization. You can set the label of a field, set its display type, and/or make it mandatory on a custom address form.



# General Development Guidelines



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides general guidelines for working with SuiteScript.

- Always thoroughly test your code before using it on your live NetSuite data.
- Type all record, field, sublist, tab, and subtab IDs in lowercase in your SuiteScript code.
- Prefix all custom script IDs and deployment IDs with an underscore (\_).
- Do not hard-code any passwords in scripts. The password and password2 fields are supported for scripting.

For additional information, see [Security Considerations](#).

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



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

- Use the built in Library functions whenever possible for reading/writing Date/Currency fields and for querying XML documents
- During script development, componentize your scripts, load them individually and then test each one -- inactivating all but the one you are testing when multiple components are tied to a single user event.
- Use **static** ID values in your API calls where applicable because name values can be changed.
- Use custom name spaces or unique prefixes for all your function names.

When working with the top-level NetSuite functions in Client SuiteScript (for example the Page Init function) it is recommended that the new function name corresponds with the NetSuite name.

For example, a Page Init function can be named pagelnit or formAPagelnit. If your code is already established, you can wrap it with a top-level function that has the appropriate naming convention.

- Since name values can be changed, ensure that you use **static** ID values in your API calls where applicable.
- Although you can use any desired naming conventions for functions within your code, it is recommended that you use custom name spaces or unique prefixes for all your function names.

When working with the top-level NetSuite functions in Client SuiteScript (for example the Page Init function) it is recommended that the new function name corresponds with the NetSuite name.

For example, a Page Init function can be named pagelnit or formAPagelnit. If your code is already established, you can wrap it with a top-level function that has the appropriate naming convention.

- 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')
```

For additional information, see [nlapiGetContext\(\)](#) and [nlobjContext](#).

- Thoroughly comment your code. This practice helps with debugging and development but and assists NetSuite support in locating problems if necessary.



# Suitelets and UI Object Best Practices



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides best practices for Suitelet development using UI objects and custom UI.

- Suitelets are ideal for generating NetSuite pages (forms, lists), returning data (XML, text), and redirecting requests.
- Limit the number of UI objects on a page (< 100 rows for sublists, < 100 options for ad-hoc select fields, < 200 rows for lists).
- Experiment with inline HTML fields embedded on [nlobjForm](#) before going the full custom HTML page route.
- Deploy Suitelets as “Available without Login” only if absolutely necessary (no user context, login performance overhead). (See [Setting Available Without Login](#).)
- Append “ifrmcntnr=T” to the external URL when embedding in iFrame especially if you are using Firefox. (See [Embed a Suitelet in iFrame](#).)
- When building custom UI outside of the standard NetSuite UI (such as building a custom mobile page using Suitelet), use the User Credentials APIs to help users manage their credentials within the custom UI. For more information, see [User Credentials APIs](#).



# User Event Best Practices



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides best practices for developing user event scripts.

- Use the `type` argument and context object to define and limit the scope of your user event logic.
- Limit the amount of script execution in user event scripts (< 5 seconds) since they run often and inline. You can use the Script Performance Monitor SuiteApp to test the performance of your scripts deployed on a specific record type. See the help topic [Application Performance Management \(APM\)](#)
- Mission critical business logic implemented using user events should be accompanied by a 'Clean up' scheduled script to account for any unexpected errors or mis-fires.
- Any operation that depends on the submitted record being committed to the database should happen in an `afterSubmit` script.
- Updating the `nlobjRecord` returned by `nlapiGetNewRecord()` in a `beforeSubmit` will affect the values that are written to the database. Updating this object in an `afterSubmit` script has no effect. This object CANNOT be submitted (because it is already being submitted).
- The `nlobjRecord` returned by `nlapiGetOldRecord()` is READ-ONLY.
- Be careful when updating transaction line items in a `beforeSubmit` script because you have to ensure that the line item totals net taxes and discounts are equal to the `summarytotal`, `discounttotal`, `shippingtotal`, and `taxtotal` amounts.
- Activities (user events) on a hosted Web site can trigger server-side SuiteScripts. In addition to Sales Orders, scripts on Case and Customer records will also execute as a result of Web activities.



# Scheduled Script Best Practices



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides best practices for working with scheduled scripts.

- It is recommended that scheduled scripts are set to run during the hours of 2 AM to 6 AM PST. Scripts set to run during the hours of 6 AM to 6 PM PST may not run as quickly due to high database activity.
- The number of **Not Scheduled** deployments to create should depend on the anticipated number of simultaneous calls you expect to make to this script and the approximate execution time of the script. A general rule of thumb is to create twice as many deployments as the total number of simultaneous calls you anticipate for this script.
- **Scheduled** deployments and **Not Scheduled** deployments are executed from the same queue. Keep this in mind as you deploy multiple scheduled scripts because your queue size is the ultimate bottle-neck for scheduled script execution throughput.
- Although there is no restriction on the number of **Not Scheduled** scripts that can be put into the NetSuite scheduling queue, too many queued scripts may create a backlog and compromise system performance. Because only one script can be run at a time, it is not recommended that you overload the system.
- If you want to deploy scheduled scripts that are **scheduled to run hourly on a 24 hour basis**, the following sample values should be set on the Script Deployment page:
  - Deployed = checked
  - Daily Event = [radio button enabled]
  - Repeat every 1 day
  - Start Date = [today's date]
  - **Start Time = 12:00 am**
  - Repeat = every hour
  - End By = [blank]
  - No End Date = checked
  - Status = Scheduled
  - Log Level = Error
  - Execute as Role = Set to **Administrator**

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



# Client Script Best Practices



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides best practices for both form-level and record-level client SuiteScript development.

- When testing form-level client scripts, use Ctrl-Refresh to clear your cache and ensure that the latest scripts are being executed.
- Global (record-level) client scripts offer a more flexible deployment model and are easier to port (bundle) than client scripts attached to forms.
- The execution of nlapiSetFieldValue and nlapiSetCurrentLineItemValue is multi-threaded whenever child field values need to be sourced in. Use the postSourcing function to synchronize your logic. Setting the *synchronous* parameter to *true* in both nlapiSetFieldValue and nlapiSetCurrentLineItem will accomplish the same thing.
- Use nlapiSetFieldValue and nlapiSetCurrentLineItemValue instead of nlapiSetFieldText and nlapiSetCurrentLineItemText if the field you are setting sometimes renders as a popup text field. Your script will execute more predictably using the **xxx Value** functions because you are setting an internal ID, which is ultimately a more precise definition. The **xxx Text** functions perform searches that may return duplicates or no results, which effectively disables your script.



# Security Considerations



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

To prevent users from accessing sensitive information such as password and credit card data, the following fields cannot be read in beforeSubmit user event scripts for external role users (for example, shoppers, online form users (anonymous users), customer center).

Field Internal ID	Field UI Label
password	Password
password2	
ccnumber	Credit Card Number
ccsecuritycode	CSC Code



# Script Optimization



**Note:** The content in this help topic pertains to all versions of SuiteScript. Be aware that currently it may only include links or examples for SuiteScript 1.0.

This section provides best practices for SuiteScript script optimization.

- As a general rule, design your user event scripts to execute in under 5 seconds, your Suitelets and Portlets to execute in under 10 seconds, and your scheduled scripts in under 5 minutes. This gives you a large enough margin of error to handle the outlier use cases (where the volume of work is unusually large, or the overall system is slow due to high load).

- Use nlapiLookupField instead of nlapiLoadRecord for fetching body field values. Note that you can fetch multiple fields at the same time.

For detailed API information, see [nlapiLookupField\(type, id, fields, text\)](#) and [nlapiLoadRecord\(type, id, initializeValues\)](#).

- Use nlapiSubmitField instead of nlapiSubmitRecord for updating body field values. Note that you can submit multiple fields at the same time.

For detailed API information, see [nlapiSubmitField\(type, id, fields, values, doSourcing\)](#) and [nlapiSubmitRecord\(record, doSourcing, ignoreMandatoryFields\)](#).

- Use inline editable child custom records whenever your use case calls for batch processing of multiple related/child records during user events on the parent record. (See [Custom Child Record Sublists](#) in the NetSuite Help Center.)

- Use nlapiScheduleScript to schedule (asynchronously execute) long-running operations from user events and Suitelets.

For detailed API information, see [nlapiScheduleScript\(scriptId, deployId, params\)](#).

- Avoid calls to [nlapiGetOldRecord\(\)](#) in user event scripts unless absolutely required (field change comparisons in afterSubmit scripts). In a beforeSubmit script you can always do a search to get current field values.

- To minimize execution logging after your script is tested and released, set your script log level to ERROR or EMERGENCY. (See [Setting Script Execution Log Levels](#).)

- Deploy scripts to run as admin only if absolutely necessary to minimize security risk and to eliminate performance overhead. (See [Executing Scripts Using a Specific Role](#).)

